Penerapan Algoritma String Matching pada Fitur *Find* Untuk Efisiensi Waktu Pencarian Informasi

Yohana Golkaria Nainggolan - 13520053
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13520053@std.stei.itb.ac.id

***Abstract*—Pembuatan makalah maupun skripsi sudah tidak asing lagi bagi seorang insan akademis, terutama ketika telah sampai di jenjang perkuliahan. Di jenjang ini, insan akademis akan banyak dituntut membuat makalah dan akan dituntut membuat skripsi sebagai syarat kelulusan. Dalam pembuatan makalah maupun skripsi, diperlukan informasi-informasi dari jurnal-jurnal maupun artikel-artikel terpercaya. Untuk efisiensi waktu pembuatan makalah, setelah artikel maupun jurnal yang dibutuhkan ditemukan, informasi yang ingin diperoleh dapat dicari dengan menggunakan *keyword* tertentu. String matching akan dilakukan sesuai dengan *keyword* yang telah diinput. Kemudian akan ditampilkan kalimat yang mengandung kata yang menjadi *keyword* biasanya dengan kata tersebut akan di-*highlight* atau di-*bold.* Pattern-nya sendiri adalah *keyword* yang diinput oleh pengguna.**

Keywords—artikel; string matching; Boyer-Moore; KnuthMoris-Path; regex

I. PENDAHULUAN

Perkembangan teknologi yang berjalan dengan sangat pesat, khususnya beberapa tahun belakangan ini, memberikan banyak kemudahan bagi manusia dalam menjalani kehidupan sehari-harinya. Hampir semua kegiatan yang dilakukan oleh manusia tidak dapat dilepaskan lagi dengan teknologi. Mulai dari membuka mata di pagi hari hingga menutup mata di malam hari, teknologi menjadi sesuatu yang sangat penting untuk digunakannya. Salah satu bentuk kemudahan yang dapat dirasakan oleh manusia adalah dalam memperoleh informasi melalui internet. Internet memberikan fasilitas yang luar biasa yang dapat menggantikan banyak barang lainnya dalam memperoleh informasi seiringnya terdapat berbagai kanal informasi elektronik yang dapat digantikan oleh internet. Ada kanal media berita elektronik yang mulai menggantikan koran, buku elektronik yang menggantikan buku fisik, dan platform media lain seperti youtube yang mulai menggantikan CD, kaset, atau bahkan televisi.

Kini masyarakat sudah lebih banyak yang menggunakan media *online* daripada media cetak sebagai sumber informasinya karena lebih mudah diakses dan tersedia 24 jam. Demikian halnya juga dengan insan akademis, dalam membuat makalah maupun skripsi sudah banyak yang memanfaatkan media *online* untuk memperoleh informasi.

Sebuah artikel maupun jurnal umumnya memiliki jumlah halaman ang tidak sedikit sehingga untuk mendapat informasi yang dibutuhkan oleh pembaca terkadang menyebabkan pembaca harus menguras waktu yang tidak sedikit. Padahal, makalah dan skripsi umumnya memiliki tuntutan batasan waktu pengumpulan tertentu. Selain itu, untuk membuatnya pun dibutuhkan banyak artikel maupun jurnal demi pembuatan makalah yang valid.

Terkadang, informasi yang dibutuhkan dari sebuah artikel maupun jurnal yang digunakan hanya sedikit, namun mengharuskan seorang pembaca membaca artikel tersebut sampai selesai. Adanya fitur *find* dalam sebuah aplikasi maupun web memudahkan pengguna dalam mendapatkan informasi yang dia butuhkan. Pengguna tidak lagi harus membaca keseluruhan dokumen untuk memperoleh informasi yang dia butuhkan. Namun pengguna dapat langsung menggunakan fitur *find* kemudian meng-*input keyword* yang dibutuhkannya. Makalah ini akan membahas metode yang dapat dilakukan untuk melakukan pencarian informasi tertentu dalam artikel menggunakan algoritma *string matching.*

Gambar 1 Contoh penggunaan fitur *find* pada artikel (Sumber: Dokumentasi Pribadi)

II. LANDASAN TEORI

A. Algoritma Pencocokan String (String Matching)

Algoritma pencocokan string (string matching) adalah algoritma yang memproses pencarian semua kemunculan string pendek P[0..n-1] yang biasa disebut sebagai pattern di string yang lebih panjang T[0..m-1] yang biasa disebut sebagai teks. Teks yaitu long string yang panjangnya n karakter. Pattern yaitu string dengan panjang m karakter (asumsi m <<< n). String adalah tipe data untuk teks yang merupakan gabungan huruf, angka, whitespace (spasi), dan berbagai karakter. Terdapat dua konsep string, yaitu suffix dan prefix. Prefix adalah huruf atau beberapa karakter (substring) yang ada di awal sebuah kata dan dapat merubah arti kata yang asli. Suffix adalah huruf atau beberapa karakter (substring) yang ada di

akhir sebuah kata dan dapat merubah arti kata yang asli. Contoh implementasi pencocokan string adalah pencarian di dalam editor teks, web search engine, analisis citra, dan bioinformatika, seperti DNA pattern matching.

Proses pencocokan string dapat dilakukan dengan beberapa algoritma berikut ini.

- Algoritma Brute Force
- Algoritma Knuth Morris Pratt (KMP)
- Algoritma Boyer-Moore (BM)
- Regular Expression

Pada pencarian informasi artikel pada editor teks maupun web ini, penulis akan menggunakan algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer-Moore (BM), dan regular expression.

B. Algoritma Knuth Morris Pratt (KMP)

Algoritma Knuth-Morris-Pratt adalah salah satu algoritma pencocokan string yang dinilai lebih baik dari algoritma Brute Force. Algoritma KMP ini dikembangkan oleh salah satu computer scientist dari Standford University, Donald E. Knuth pada tahun 1967. Setahun sebelumnya, James H. Morris dan Vaughan R. Pratt juga mengembangkan algoritma yang sama, oleh karena itu nama dari algoritma ini diambil dari nama ketiga penciptanya, yakni Knuth, Morris, dan Pratt.

Pada dasarnya, algoritma ini mirip seperti algoritma Brute Force dalam pencocokan string, yaitu melakukan pencocokan dari kiri ke kanan string. Namun ada hal yang berbeda. Pada algoritma Brute Force, pergeseran dalam pencocokan string dilakukan satu persatu dari kiri ke kanan, baik karakter yang dicocokkan sama atau berbeda. Pergeseran satu persatu dilakukan hingga pencocokan karakter sudah mencapai akhir atau ditemukan string yang dicari oleh pattern pada teks. Berbeda dengan algoritma Brute Force, algoritma KMP melakukan pergeseran dalam pencocokan string dengan mempertimbangkan pattern yang telah dibandingkan. Sehingga pada algoritma KMP dibutuhkan pre-processing untuk menentukan jumlah pergeseran yang dilakukan untuk setiap indeks pada pattern atau yang disebut dengan fungsi pinggiran.

Fungsi pinggiran atau failure function pada algoritma KMP adalah fungsi yang menghitung jumlah pergeseran yang dilakukan dengan menghitung panjang terbesar prefiks dari pattern, P[0..k], yang merupakan sufiks dari pattern tersebut, P[1..k], dengan k = (indeks mismatch pada pattern) - 1. Dengan demikian, proses pergeseran akan menjadi lebih efisien karena karakter yang sudah dicocokkan pada prefiks tidak dicocokkan kembali. Algoritma KMP menghasilkan kompleksitas waktu O(m+n) dengan O(m) adalah kompleksitas menghitung fungsi pinggiran dan O(n) adalah kompleksitas pencarian string.

Sebagai contoh, pada pattern T = ‘abcabd’, jika terjadi ketidakcocokan pada d (j = 5), cari suffix dari T[0..j-1] yang juga merupakan prefix dari T[0..j-1]. Dalam kasus ini, prefix ‘ab’ (T[0..1]) sama dengan suffix ‘ab’ (T[3..4]) . Kita tahu bahwa prefix ‘ab’ sudah cocok, maka geser T sehingga prefix ‘ab’ ada di posisi suffix ‘ab’ dengan menggeser T sejauh 3 karakter.

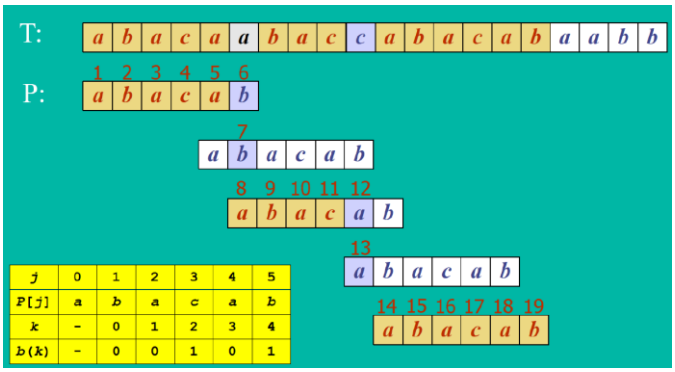
Contoh:

i : 012345678
S : abcabcabd
T : abcabd
j : 012345

Terjadi ketidakcocokan pada d(j=5). Geser T sehingga prefix 'ab' ada di posisi suffix 'ab'

i : 012345678
S : abcabcabd
T : abcabd
j : 012345

Kita sudah tahu bahwa T[0..1] cocok dengan S[3..4], sehingga pencocokan dimulai dari j = 2. Pada akhirnya ditemukan bahwa T cocok dengan S di i = 3.



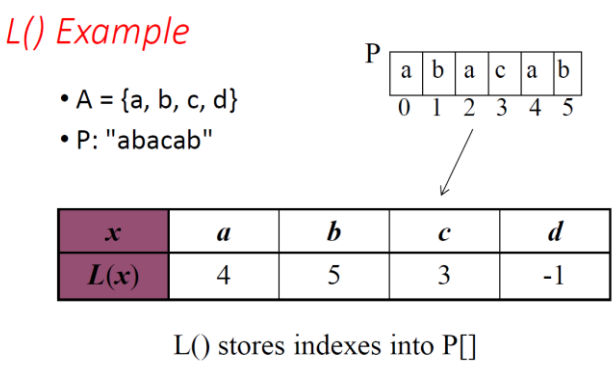
Gambar 2 Pencocokan string dengan algoritma KMP
(Sumber: Bahan Kuliah IF2211 Strategi Algoritma, Pencocokan String (String/Pattern Matching))

C. Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan string dengan teknik yang cukup unik namun dinilai paling efisien. Algoritma Boyer-Moore menggunakan 2 teknik, yaitu *looking-glass technique* dan *character-jump technique*. Teknik *looking-glass* ini akan mencocokkan *pattern* dengan teks dengan pengecekan mundur. Pengecekan mundur seperti ini efisien karena dapat memprediksi pergeseran yang akan dilakukan selanjutnya secara optimal. Lalu teknik *character-jump* adalah teknik pergeseran yang dilakukan pada algoritma Boyer-Moore dengan mempertimbangkan lokasi kemunculan karakter yang *mismatch* antara *pattern* dengan teks. Terdapat tiga kasus dalam teknik *character-jump* ini, yaitu karakter yang *mismatch* pada teks ditemukan di sebelah kiri karakter *mismatch* pada *pattern*, karakter yang *mismatch* pada teks ditemukan di sebelah kanan karakter *mismatch* pada *pattern*, dan karakter yang *mismatch* pada teks tidak terdapat pada *pattern*.

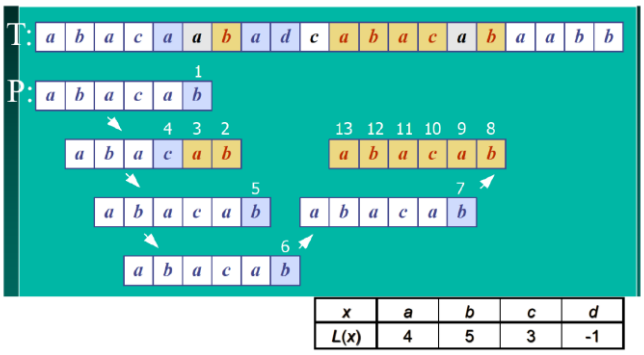
Sama seperti algoritma Knuth-Morris-Pratt, algoritma Boyer-Moore juga membutuhkan pre-processing untuk mendapatkan lokasi kemunculan karakter yang ada pada teks. Pre-processing pada algoritma ini adalah digunakan untuk

mencari lokasi *last occurance* dari setiap karakter pada teks. Jika karakter ada pada *pattern*, maka nilai *last occurrence*-nya adalah indeks terbesar kemunculan karakter tersebut pada *pattern* P, namun jika tidak ditemukan, maka nilai last occurancenya adalah -1. Contoh mencari nilai *last occurance* adalah sebagai berikut :



Gambar Fungsi *Last Occurance* pada algoritma Boyer-Moore (Sumber: Bahan Kuliah IF2211 Strategi Algoritma, Pencocokan *String*)

Last occurance ini berguna untuk menentukan berapa banyak pergeseran yang akan dilakukan. Misalkan *mismatch* terjadi pada karakter X. Karakter X akan dicek apakah ada pada *pattern* menggunakan *last occurance*. Jika berada di sebelah kiri indeks karakter *mismatch*, maka *pattern* digeser ke kanan sehingga karakter yang *mismatch* tadi dapat benar dicocokkan dengan karakter pada *pattern*. Berikut adalah contoh pencocokan *string* dengan algoritma Boyer-Moore :

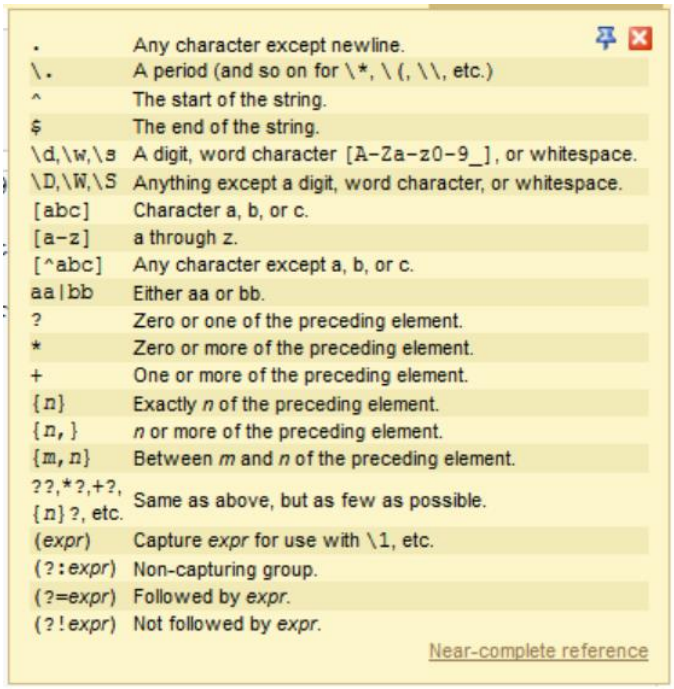


Gambar 4 Pencocokan *string* dengan algoritma Boyer-Moore (diambil dari Bahan Kuliah IF2211 Strategi Algoritma, Pencocokan *String*)

D. Regular Expression (Regex)

Regular Expression atau biasa disebut regex adalah salah satu metode pencocokan string yang unggul. Metode regex dinilai unggul karena dapat melakukan pencarian dengan banyak pola sekaligus. Misalkan saja ingin mencari kata berawalan "Sa", memiliki huruf "ra", kata berakhiran "GA", atau bahkan mencocokkan penulisan tanggal. Pencarian dalam regex lebih menekankan pada pola pencarian yang diberikan, sehingga banyak modifikasi pola-pola yang dapat dilakukan. Beberapa bahasa pemrograman sudah menyediakan library regex, seperti pada Python, Javascript, Java, dan PHP. Bahkan ada beberapa situs yang menyediakan pencocokan string menggunakan metode regex, seperti regextester.com dan pythex.org. Untuk memahami regex cukup sulit, karena harus mencari pola yang benar-benar ingin dicari. Ditambah lagi penulisan pola pada regex cukup kompleks karena menggunakan simbol-simbol yang sebelumnya cukup jarang ditemukan.

Pada algoritma KMP dan BM, masukan kata atau pattern yang dicari haruslah eksak, sehingga hasil yang diberikan hanya kata-kata yang sesuai dengan pattern masukan. Namun pada metode regex, pattern yang dimasukkan dapat berupa pattern yang eksak atau pola yang ingin dicari. Misalkan pola yang dimasukkan adalah '^([01][0-9] | 2[0-3]):[0-5][0-9]:[0-5][0-9]'. Maka regex akan mencari semua kata yang terdapat pada text yang sesuai dengan pola di atas. Contoh teks yang sesuai adalah 23:59:59 atau 06:15:05. Keduanya merupakan hasil yang benar, karena masukannya berupa pola, bukan pattern yang eksak.



Gambar 5 Notasi Umum Regex (diambil dari Bahan Kuliah IF2211 Strategi Algoritma, *String Matching dengan Regex*)

III. IMPLEMENTASI DAN PENGUJIAN

Dalam penerapannya, sebelum melakukan *string matching*, pengguna harus terlebih dahulu memasukkan *keyword* yang ingin dicari. Setelah itu, fitur *find* akan melakukan proses *string matching* dan setelah dilakukan proses *string matching*, akan ditampilkan *keyword* yang dicari dimana umumnya *keyword* tersebut akan di-*bold* atau di-*highlight*. Pada kasus ini, *keyword* akan menjadi *pattern* dan keseluruhan isi artikel akan menjadi teks. Implementasi dilakukan dengan menggunakan bahasa *python*.

A. Implementasi dengan Menggunakan Algoritma Knuth-Morris-Pratt (KMP)

T: (artikel atau jurnal)
P: ekonomi

Fungsi pinggiran,

- Jika j = 0, maka tidak ada indeks yang dimulai dari 0 dan berakhir pada -1. Sehingga nilai k dan b(k) untuk j = 0 tidak ada atau bisa ditandai dengan strip "-".
- Jika j = 1 dan k = 1 – 1 = 0, maka prefix P[0…0] adalah "e" namun tidak ada suffix yang dapat dibentuk dari pattern P[1…0], maka b(k) bernilai 0 karena tidak ada prefix yang juga merupakan suffix dari pattern P.
- Jika j = 2 dan k = 2 – 1 = 1, maka prefix dari pattern P[0..1] = "ek" adalah "e" dan "ek", suffix dari P[1..1] = "k" adalah "k", sehingga tidak ada prefix dan juga merupakan suffix dari P, mengakibatkan b(k) bernilai 0.
- Jika j = 3 dan k = 3 – 1 = 2, maka prefix dari pattern P[0..2] = "eko" adalah "e", "ek" dan "eko", suffix dari P[1..2] = "ko" adalah "o" dan "ko", sehingga tidak ada prefix dan juga merupakan suffix dari P, mengakibatkan b(k) bernilai 0.
- Jika j = 4 dan k = 4 – 1 = 3, maka prefix dari pattern P[0..3] = "ekon" adalah "e", "ek", "eko" dan "ekon", suffix dari P[1..3] = "kon" adalah "n", "on" dan "kon", sehingga tidak ada prefix dan juga merupakan suffix dari P, mengakibatkan b(k) bernilai 0.
- Jika j = 5 dan k = 5 – 1 = 4, maka prefix dari pattern P[0..4] = "ekono" adalah "e", "ek", "eko", "ekon" dan "ekono", suffix dari P[1..4] = "kono" adalah "o", "no", "ono" dan "kono", sehingga tidak ada prefix dan juga merupakan suffix dari P, mengakibatkan b(k) bernilai 0.
- Jika j = 6 dan k = 6 – 1 = 5, maka prefix dari pattern P[0..5] = "ekonom" adalah "e", "ek", "eko", "ekon", "ekono" dan "ekonom", suffix dari P[1..5] = "konom" adalah "m", "om", "nom", "onom" dan "konom", sehingga tidak ada prefix dan juga merupakan suffix dari P, mengakibatkan b(k) bernilai 0.

j	0	1	2	3	4	5	6
P[j]	e	k	o	n	o	m	i

Keyword tersebut akan dicari di keseluruhan artikel dan akan ditampilkan dengan cara di-*highlight* atau di-*bold*. (Contoh konkret dari penggunaan algoritma Knuth-Morris-Pratt tidak dapat ditampilkan karena artikel yang cukup panjang).

- Algoritma KMP

```
class kmp :
    def __init__(self):
        pass

    def failureFunction(self, pattern) :
        failure = [0] * len(pattern)
        m = len(pattern)
        i = 1
        j = 0

        while (i < m) :
            if (pattern[i] == pattern[j]) :
                failure[i] = j + 1
                i += 1
                j += 1
            elif (j > 0) :
                j = failure[j-1]
            else :
                failure[i] = 0
                i += 1
        return failure

    def match(self, text, pattern) :
        m = len(pattern)
        n = len(text)
        i = 0
        j = 0
        failure = self.failureFunction(pattern)

        while (i < n) :
            if (text[i] == pattern[j]) :
                if (j == (m - 1)) :
                    return i - m + 1
                i += 1
                j += 1
            elif (j > 0) :
                j = failure[j-1]
            else :
                i += 1

        return -1
```

B. Implementasi dengan Menggunakan Algoritma BM

Algoritma Boyer-Moore sangat efisien dan lebih cepat daripada algoritma Brute Force dan Knuth-Morris-Pratt dalam beberapa kasus, terutama bila karakter-karakter yang ada di dalam teks dan *pattern* beragam yang mengakibatkan banyak sekali nilai L(x) bernilai -1 sehingga jika ditemukan *mismatch* pada karakter di indeks j dan karakter tersebut tidak terdapat di dalam *pattern* (L(x) = -1) maka *pattern* akan langsung digeser sehingga karakter paling kiri *pattern* sejajar dengan karakter teks pada indeks ke-(j+1) atau karakter pada K[j+1].

- Algoritma BM

```
class bm :
    def __init__ (self) :
        pass

    def lastOccuranceFunction (self, pattern) :
        lastOccurance = [-1]*256
        for i in range (len(pattern)) :
            lastOccurance[ord(pattern[i])] = i
        return lastOccurance

    def match(self, text, pattern) :
        m = len(pattern)
        n = len(text)
        i = m - 1
        lastOccurance = self.lastOccuranceFunction(pattern)

        if (i > (n - 1)) :
            return -1

        j = m - 1
        while (i <= (n - 1)):
            if (text[i] == pattern[j]) :
                if (j == 0) :
                    return i
                else :
                    i -= 1
                    j -= 1
            else :
                lo = lastOccurance[ord(text[i])]
                i = i + m - min(j, lo + 1)
                j = m - 1

        return -1
```

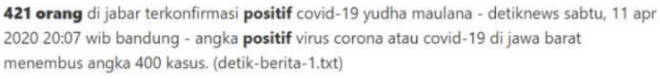
C. Implementasi dengan Menggunakan Regex

- Regex

```
import re

class regex :

    def __init__(self) :

        pass

    def match(self, text, pattern) :

        x = re.search(pattern, text)

        if (x != None) :

            return x.start()

        return -1
```

D. Hasil Uji Coba Program

Untuk uji coba 1, *keyword* yang digunakan adalah positif. Hasilnya sebagai berikut:



Gambar 6 Hasil Luaran Program dengan *keyword* positif
Sumber (Dokumen Pribadi)

Pada hasil uji coba di atas, dapat dilihat bahwa kata positif di-bold yang mengindikasikan kata tersebut merupakan *keyword* yang dicari.

Untuk uji coba yang kedua, *keyword* yang digunakan masih sama yaitu positif, namun menggunakan artikel yang tidak memiliki kata positif di dalamnya. Hasil yang didapat adalah sebagai berikut:

Gambar 7 Hasil Luaran Program Ketika Tidak Mengandung *keyword* yang diinginkan
Sumber (Dokumen Pribadi)

Pada hasil uji coba di atas, dapat dilihat bahwa kata positif tidak ditemukan di dalam artikel.

IV. SIMPULAN

Pencocokan string merupakan suatu algoritma yang digunakan untuk mencari lokasi pertama dalam teks yang bersesuaian dengan pattern. Ada beberapa macam algoritma pencocokan string, yaitu algoritma Brute Force, algoritma Knuth-Morris-Pratt (KMP), algoritma Boyer Moore dan Regular Expression. Algoritma pencocokan string (string matching) dapat digunakan dalam kehidupan sehari-hari dan memberi kemudahan bagi manusia dalam menjalani kehidupannya. Salah satu implementasi pencocokan string yang sering kita temukan dan gunakan adalah pada fitur *find* yang umumnya ada pada text editor maupun web. Fitur ini sering digunakan untuk mencari informasi tertentu yang dibutuhkan pengguna dari sebuah teks yang dapat dikatakan cukup panjang. Fitur *find* akan memberi *highlight* untuk kata yang sesuai dengan

keyword Fitur *find* akan memberi *highlight* untuk kata yang sesuai dengan *keyword* yang dimasukkan oleh pengguna. Hasil penelitian dari makalah ini tentu masih banyak memiliki kekurangan, karena keterbatasan penulis dalam membuat program untuk fitur *find*. Penelitian ini masih dapat mengalami perkembangan dan perbaikan lagi untuk menuju hasil yang diperoleh dapat menjadi lebih baik lagi.

V. Ucapan Terima Kasih

Puji syukur penulis panjatkan ke hadirat Tuhan Yang Maha Esa karena atas berkat dan rahmatNya-lah, penulis dapat menyelesaikan penulisan makalah ini dengan tepat waktu. Penulis juga mengucapkan terima kasih kepada para Bapak/Ibu Dosen pengampu mata kuliah IF2211 Strategi Algoritma : Ibu Dr. Masayu Leylia Khodra, Ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc., dan Bapak Dr. Ir. Rinaldi Munir, M.T., yang telah mengajar mata kuliah IF2211 selama satu semester ini dan telah membantu penulis untuk memahami materi yang dijadikan sebagai bahan acuan dalam pembuatan makalah ini. Tak lupa pula, penulis mengucapkan terima kasih kepada semua pihak yang tidak dapat penulis sebutkan satu persatu yang telah berkontribusi baik secara langsung maupun tidak langsung dalam membantu penyelesaian makalah ini.

References

[1] Program Studi Teknik Informatika. 2021. Pencocokan String (String/Pattern Matching). Diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. Diakses tanggal 18 Mei 2022.

[2] GeeksforGeeks. 2020. Applications of String Matching Algorithms. Diambil dari https://www.geeksforgeeks.org/applications-of-stringmatching-algorithms/. Diakses tanggal 18 Mei 2021.

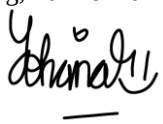
[3] GeeksforGeeks. 2021. KMP Algorithm for Pattern Searching. Diambil dari https://www.geeksforgeeks.org/kmp-algorithm-for-patternsearching/. Diakses tanggal 20 Mei 2021.

[4] Program Studi Teknik Informatika. 2021. String Matching dengan Regex. Diambil dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf. Diakses tanggal 18 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, 20 Mei 2022



Yohana Golkaria Nainggolan - 13520053