

Aplikasi Algoritma Boyer-Moore dan *Regular Expression* pada Pencarian Rima Kata

Muhammad Dehan Al Kautsar / 13519200
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519200@std.stei.itb.ac.id

Abstract—Teknologi keinformatikaan berkembang secara cepat sehingga banyak sekali kegiatan manusia yang dapat dibantu oleh otomasi program. Salah satunya adalah pencarian rima kata untuk membantu seseorang dalam membuat sastra seperti puisi ataupun pantun. Pencarian rima kata adalah mencari rima kata di dalam suatu *pattern* untuk dicari kembali kata-kata apa saja di Kamus Besar Bahasa Indonesia yang memiliki rima kata yang sesuai dengan *pattern* tersebut. Pencarian rima kata dapat dilakukan dengan berbagai cara, namun pada makalah ini akan ditunjukkan pencarian rima kata tersebut menggunakan aplikasi algoritma Boyer-Moore dan Regular Expression.

Keywords—Rima kata, *String Matching*, *Boyer-Moore Algorithm*, *Regular Expression*

I. PENDAHULUAN

Dewasa ini, dapat dilihat bahwa budaya sastra dan literatur berkembang pesat dan banyak diminati terutama oleh generasi muda. Puisi dan pantun menjadi hal yang cukup sering didengar oleh telinga kita, dan puisi serta pantun yang elok dan indah didengar dapat menjadi hal yang sangat menarik dalam benak kita.

AJAKAN*

Ida
Menembus sudah caya
Udara tebal kabut
Kaca hitam lumut
Pecah pecar sekarang
Di ruang legah lapang
Mari ria lagi
Tujuh belas tahun kembali
Bersepeda sama gandengan
Kita jalani ini jalan

Ria bahagia
Tak acuh apa-apa
Gembira girang
Biar hujan datang
Kita mandi-basahkan diri
Tahu pasti sebentar kering lagi.

Februari 1943

Gambar 1. Salah Satu Puisi Karya Chairil Anwar
(Sumber: <https://titikdua.net/puisi-chairil-anwar/> diakses pada 8 Mei 2021 pukul 19.14 WIB)

Namun dalam membuat sastra seperti puisi ataupun pantun membutuhkan keterampilan serta kosakata yang cukup banyak sehingga tidak semua manusia dapat menciptakan puisi ataupun pantun yang cukup indah untuk didengarkan di khalayak ramai.

Di lain sisi, perkembangan ilmu pengetahuan terutama di bidang komputerisasi semakin pesat dan menarik untuk diperhatikan perkembangannya. Berkaitan dengan permasalahan yang ada diatas, timbul sebuah solusi yang dapat memudahkan tiap orang untuk memudahkan membuat sebuah karya sastra layaknya puisi ataupun pantun dengan memperhatikan rima kata tiap kalimatnya menggunakan pendekatan *string matching*.

II. LANDASAN TEORI

A. Pencocokan String

String adalah tipe data berupa kumpulan karakter yang panjangnya tidak terbatas. Dalam bahasa pemrograman, string disebut dengan *array of character* meskipun karakter tersebut tidak sepenuhnya harus berupa karakter namun dapat berupa angka maupun karakter khusus.

Apabila kita mempunyai string S dengan panjang N, maka:

$$S = X_0X_1 \dots X_{N-1}$$

Dengan $X[i]$, dan $0 \leq i \leq N-1$, merupakan karakter pada posisi ke-i.

Pencocokan string (*string matching*) sendiri adalah pencocokan antara kedua string untuk mencari kesamaan di dalamnya. String pertama adalah *text*, yaitu *long string* berukuran n karakter. Sedangkan string kedua adalah *pattern* yang merupakan *short string* dengan panjang m karakter (asumsi $m \ll n$) yang akan dicari di dalam *text*. Konsep pencocokan string adalah mencari dimanakah lokasi pertama ditemukannya *pattern* di dalam *text* tersebut.

Pencocokan string dapat dilakukan dengan menggunakan beberapa teknik/algoritma, beberapa diantaranya adalah:

1. Algoritma Brute Force,
2. Algoritma Knuth Morris Pratt,
3. Algoritma Boyer-Moore,

4. Algoritma Metaphone,
5. dan lain-lain.

B. Algoritma Boyer-Moore

Seperti yang sudah disebutkan sebelumnya bahwa algoritma Boyer-Moore merupakan salah satu algoritma yang cukup efisien dalam melakukan pencocokan string. Berbeda dengan beberapa metode pencocokan string lainnya, algoritma ini melakukan pengecekan string *pattern* dimulai dari bagian belakang *pattern* untuk mendapatkan beberapa info yang tidak dimiliki pencocokan string yang dilakukan dari kiri *pattern*. Algoritma Boyer-Moore memiliki dua teknik penting, yaitu:

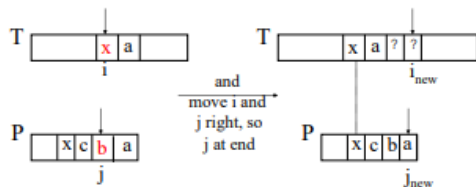
1. *Looking Glass Technique*

Teknik ini dilakukan dengan cara mencari *pattern* dalam *text* secara *backward* melalui indeks terakhir dari *pattern*.

2. *Character Jump Technique*

Ketika melakukan *Looking Glass Technique*, dan diketahui bahwa $T[i] \neq P[j]$, dan dimisalkan $T[i]$ bernilai x , maka terdapat tiga kasus:

Kasus 1: Jika di dalam *pattern* juga terdapat x , dan *last occurrence* dari x terdapat pada $k < j$, maka yang perlu dilakukan adalah melakukan *shift right* sehingga $T[i]$ sejajar dengan $P[k]$.

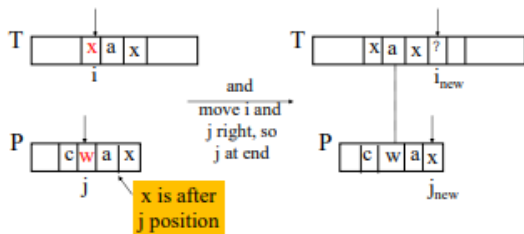


Gambar 2. Kasus 1 Character Jump Technique

(Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf> diakses pada 8 Mei 2021 21.17 WIB)

Kasus 2: Jika di dalam *pattern* juga terdapat x , dan *last occurrence* dari x terdapat pada $k > j$, maka yang perlu dilakukan adalah melakukan *shift right* sebanyak satu kali.

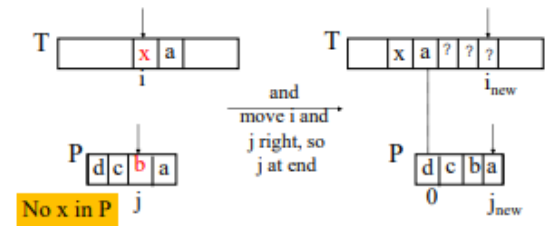


Gambar 3. Kasus 2 Character Jump Technique

(Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf> diakses pada 8 Mei 2021 21.17 WIB)

Kasus 3: Jika di dalam *pattern* tidak terdapat x , maka yang dilakukan adalah menggeser *pattern* sehingga $P[0]$ sejajar dengan $T[i+1]$



Gambar 4. Kasus 3 Character Jump Technique

(Sumber:

<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf> diakses pada 8 Mei 2021 21.17 WIB)

Sebagai contoh, kita ingin mencari pola “CAGT” dalam rantai DNA “TGACTGACAGTA”, jika kita menggunakan algoritma boyer-moore maka akan dicari terlebih dahulu *last occurrence* dari tiap-tiap karakter *pattern*nya

x	A	C	G	T
L(x)	1	0	2	3

Selanjutnya algoritma akan melakukan pencocokan string dengan urutan perbandingan sebagai berikut:

T	G	A	C	T	G	A	C	A	G	T	A
			1								
C	A	G	T								

Karena $T[3] \neq P[3]$, maka *pattern* akan digeser sehingga $T[3]$ sejajar dengan *last occurrence* dari $T[3]$ di dalam *pattern* yaitu $P[0]$ (kasus 2).

T	G	A	C	T	G	A	C	A	G	T	A
						2					
			C	A	G	T					

Karena $T[6] \neq P[3]$, maka *pattern* akan digeser sehingga $T[6]$ sejajar dengan *last occurrence* dari $T[6]$ di dalam *pattern* yaitu $P[1]$ (kasus 2).

T	G	A	C	T	G	A	C	A	G	T	A
								3			
					C	A	G	T			

Karena $T[8] \neq P[3]$, maka *pattern* akan digeser sehingga $T[8]$ sejajar dengan *last occurrence* dari $T[8]$ di dalam *pattern* yaitu $P[1]$ (kasus 2).

T	G	A	C	T	G	A	C	A	G	T	A
							7	6	5	4	
							C	A	G	T	

Setelah dilakukan pengecekan dari belakang, didapatkan bahwa ditemukan kecocokan antara *text* dengan *pattern*-nya yaitu di indeks ke-7 dengan 7 kali perbandingan karakter.

C. Regular Expression

Regular Expression adalah salah satu metode dalam melakukan pencarian pola string yang mengikuti beberapa kaidah dan pola tertentu. Pola tersebut biasanya digunakan dalam algoritma pencarian string dan merupakan teknik yang dikembangkan dalam bidang ilmu komputer dan teori bahasa formal otomatis.

Beberapa pola *regular expression* yang nantinya akan berguna dalam pencocokan string antara lain:

Notasi	Makna
.	Semua karakter kecuali <i>newline</i>
\.	Titik (yaitu sebuah <i>meta-character</i> dan berlaku pula untuk *, \(), \, dan sebagainya)
^	Awalan string
\$	Akhiran string
\d, \w, \s	Sebuah digit, <i>word</i> , dan <i>whitespaces</i> (spasi)
\D, \W, \S	Semuanya kecuali digit, <i>word</i> , dan <i>whitespaces</i> (spasi)
[abc]	Karakter a,b, atau c
[a-z]	Karakter a hingga z
[0-9]	karakter 0 hingga 9
[^abc]	Seluruh karakter kecuali a,b,dan c
?	0 atau 1 dari karakter sebelumnya
+	1 atau lebih dari karakter sebelumnya
*	0 atau lebih dari karakter sebelumnya
{n}	n buah karakter sebelumnya
{m,n}	Diantara m dan n buah karakter sebelumnya
(expr)	Menangkap ekspresi di tanda kurung tersebut.

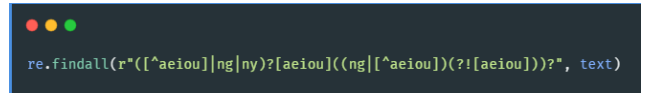
D. Rima Kata

Menurut Kamus Besar Bahasa Indonesia, rima kata bermakna pengulangan bunyi yang berselang, baik di dalam larik sajak maupun pada akhir laris sajak yang berdekatan. Sehingga dapat dilihat bahwa dalam beberapa kasus tiap larik sajak yang dapat berupa puisi, pantun, gurindam, dan lain-lainnya memiliki perulangan di tiap lariknya (biasanya di tiap

suffix kalimatnya). Hal ini pula yang menambah keindahan di tiap sajaknya karena keunikan dari hal tersebut.

III. APLIKASI REGULAR EXPRESSION DAN BOYER-MOORE PADA PENCARIAN RIMA KATA

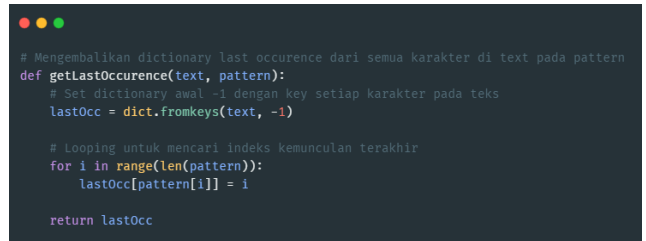
Secara garis besar, *regular expression* dan algoritma Boyer-Moore akan sangat berguna di dalam pencarian rima kata. Sebagai contoh permasalahan, terdapat seorang mahasiswa bernama Dehan yang ingin membuat sebuah pantun 4 larik. Ia ingin mencari sebuah kata yang memiliki rima kata yang sama dengan kata “senja” untuk digunakan di dalam sampiran pantun. Maka yang dapat dilakukan pertama kali adalah memanfaatkan *regular expression* pada kata tersebut.



Gambar 5. Regular Expression untuk Mencari Suku Kata (Sumber: Dokumen Pribadi)

Regular expression pertama-tama digunakan untuk mencari tiap suku kata dalam kata tersebut. Dengan contoh kata “senja” diatas, suku kata dari kata “senja” dapat dicari dengan *regular expression* pada gambar 5 di atas sehingga akan menghasilkan suku kata “sen” dan “ja”. Setelah didapatkan suku kata-nya, maka program akan mengambil suku kata terakhir dari *pattern* tersebut, untuk mengecek kecocokan *suffix* pada *pattern* tersebut dengan kata-kata yang terdapat pada kamus program.

Untuk melakukan pencocokan *pattern suffix* tersebut dengan kamus program, digunakan algoritma Boyer-Moore sebagai algoritma pencocokan string (*string matching*). Sebelum melakukan aplikasi algoritma Boyer-Moore, berikut adalah algoritma untuk melakukan pencarian *last occurrence* dari sebuah huruf di dalam *pattern* dalam bahasa Python.



Gambar 6. Algoritma Mencari Last Occurrence (Sumber: Dokumen Pribadi)

Setelah algoritma mencari *last occurrence* dibuat untuk melakukan pencarian huruf terakhir yang muncul di dalam *pattern*, baru lah dibuat pula aplikasi algoritma Boyer-Moore untuk mencari *pattern* “ja” di dalam tiap kata di dalam kamus program.

```

# Mengembalikan index awal kemunculan pattern atau -1 jika tidak ditemukan
def boyerMoore(text, pattern):
    # Ubah ke lowercase agar tidak case sensitive
    textLower = text.lower()
    patternLower = pattern.lower()

    # Cari index kemunculan terakhir alfabet di pattern
    lastOccurrence = getLastOccurrence(textLower, patternLower)

    # Hitung panjang teks dan pattern
    lenT = len(textLower)
    lenP = len(patternLower)

    # Panjang pattern lebih besar dari panjang teks (Tidak ditemukan kecocokan)
    if lenP > lenT:
        return -1

    # Set pencarian awal dari indeks paling akhir pattern
    i = lenP - 1
    j = lenP - 1

    # Looping sepanjang teks
    while (i < lenT):
        # Jika karakter pada pattern dan teks sama
        if (textLower[i] == patternLower[j]):
            # Jika kesamaan ditemukan pada indeks pertama
            # Pattern ditemukan pada teks
            if (j == 0):
                return i
            # Jika tidak, periksa karakter selanjutnya
            else:
                i += 1
                j += 1
        # Karakter pada pattern dan teks berbeda
        else:
            # Ambil index kemunculan terakhir
            lastCharIdx = lastOccurrence[textLower[i]]

            # Hitung nilai i baru
            i += lenP
            # Jika index kemunculan terakhir lebih kecil dari index ketidakcocokan
            if lastCharIdx < j:
                i = lastCharIdx + 1
            # Jika index kemunculan terakhir lebih besar dari index ketidakcocokan
            elif lastCharIdx > j:
                i = j
            # Jika index kemunculan = -1, nilai i baru = (i + panjang pattern)

        # Ulangi pencarian pada index terakhir pattern
        j = lenP - 1

    # Tidak ditemukan kecocokan
    return -1

```

Gambar 7. Algoritma Boyer-Moore
(Sumber: Dokumen Pribadi)

Algoritma Boyer-Moore menggunakan fungsi *last occurrence* untuk membantu algoritma tersebut untuk keperluan *shift right pattern* tersebut terhadap *text*. Apabila tidak ditemukannya *pattern* di dalam *teks*, maka algoritma akan mengembalikan nilai -1, sedangkan apabila ditemukannya *pattern* tersebut algoritma akan mengembalikan indeks awal ditemukannya *pattern* tersebut.

Misalkan saja di dalam kamus program terdapat kata “kemboja” dan “menjanggal”. Maka yang dilakukan oleh algoritma Boyer-Moore adalah sebagai berikut:

Tabel *last occurrence* suku kata terakhir kata “senja”:

x	j	a
L(x)	0	1

1. Kemboja

k	e	m	b	o	j	a
	1					
j	a					

k	e	m	b	o	j	a
			2			
		j	a			

k	e	m	b	o	j	a
					3	
				j	a	

k	e	m	b	o	j	a
					5	4
				j	a	

Karena *pattern* juga ditemukan di akhir kata, maka kata “kemboja” menjadi rima kata dari kata “senja” dan akan diberikan kembali kepada penggunanya.

2. Menjanggal

m	e	n	j	a	n	g	g	a	l
	1								
j	a								

m	e	n	j	a	n	g	g	a	l
			2						
		j	a						

m	e	n	j	a	n	g	g	a	l
			4	3					
			j	a					

Dapat dilihat bahwa *pattern* ditemukan di tengah kata, sehingga *text* (menjanggal) tersebut bukanlah kata yang mempunyai rima kata yang sama dengan “senja”.

Algoritma ini akan sangat berguna apabila pengguna memiliki kamus data yang sangat lengkap dan banyak sehingga kata apapun yang dimasukkan sebagai *input* pengguna, algoritma ini akan memberikan solusi kata apa saja yang memiliki rima kata yang sama dengan *input* dari pengguna tersebut.

IV. ANALISIS PROGRAM DAN KASUS

```
import re

# Mengembalikan dictionary last occurrence dari semua karakter di text pada pattern
def getLastOccurence(text, pattern): ...

# Mengembalikan index awal kemunculan pattern atau -1 jika tidak ditemukan
def boyerMoore(text, pattern):
    # Ubah ke Lowercase agar tidak case sensitive
    textLower = text.lower()
    patternLower = pattern.lower()

    # Cari index kemunculan terakhir alfabet di pattern
    lastOccurence = getLastOccurence(textLower, patternLower)

    # Hitung panjang teks dan pattern
    lenT = len(textLower)
    lenP = len(patternLower)

    # Panjang pattern Lebih besar dari panjang teks (Tidak ditemukan kecocokan)
    if lenP > lenT:
        return -1

    # Set pencarian awal dari indeks paling akhir pattern
    i = lenP - 1
    j = lenP - 1

    # Looping sepanjang teks
    while (i < lenT):
        # Jika karakter pada pattern dan teks sama
        if (textLower[i] == patternLower[j]):
            # Jika kesamaan ditemukan pada indeks pertama
            # Pattern ditemukan pada teks

            if (j == 0):
                return i
            # Jika tidak, periksa karakter selanjutnya
            else:
                i += 1
                j += 1
            # Karakter pada pattern dan teks berbeda
            else:
                # Ambil indeks kemunculan terakhir
                lastCharIdx = lastOccurence[patternLower[j]]

                # Hitung nilai i baru
                i += lenP

                # Jika indeks kemunculan terakhir Lebih kecil dari indeks ketidakcocokan
                if lastCharIdx < j:
                    i = lastCharIdx + 1
                # Jika indeks kemunculan terakhir Lebih besar dari indeks ketidakcocokan
                elif lastCharIdx > j:
                    i = j
                # Jika indeks kemunculan = -1, nilai i baru = (i + panjang pattern)

                # Ulangi pencarian pada indeks terakhir pattern
                j = lenP - 1

    # Tidak ditemukan kecocokan
    return -1

# Memeriksa apakah kata dalam keywords ada dalam text
# all = True -> semua kata dalam keywords
# all = False -> minimal salah satu kata dalam keywords
def textContains(text, keywords, all=False):
    index = boyerMoore(text, keywords)
    if index != -1:
        if index == 0:
            return True, index
        return False, -1

# Mendapatkan list of syllable dari suatu pattern
def getSyllable(pattern):
    return re.findall(r"([^\aeiou]nglmy)[aeiou]((ngl[^\aeiou])?){1}[aeiou])?", pattern)

def addToResult(pattern, kamus, result):
    for el in kamus:
        found, index = textContains(el, pattern)
        if (found):
            if (index == len(el)-len(pattern)):
                if (inputWord != el):
                    result.append(el)
            else:
                #kasus awalan dan akhiran mempunyai hal yang sama dengan pattern
                addToResult(pattern, [el[index:len(pattern):]], result)

kamus = ["kemboja", "sahaja", "menjanggal", "larang", "sekarang", "menangis", "gandengan"]

inputWord = input("Kata yang ingin dicari: ")
syllables = getSyllable(inputWord)
apattern adalah suku kata yang ingin dibandingkan dengan kamus
pattern = syllables[len(syllables)-1][0]
result = []

addToResult(pattern, kamus, result)

print("Kata yang memiliki rima kata yang sama:")
for el in result:
    print(el)
```

Gambar 8. Program Mencari Rima Kata (Sumber: Dokumen Pribadi)

Program di atas adalah program yang dapat mencari kata-kata yang memiliki rima kata yang sama dengan *input* dari penggunaannya. Untuk kamus programnya, penulis menggunakan beberapa kata *dummy* untuk memudahkan pengerjaan program, namun alangkah lebih baik apabila pengguna memiliki sumber kamus tersendiri untuk memperkaya kosakata program untuk mencari kata-kata yang memiliki rima kata yang sama dengan *input* kata.

Dari segi kompleksitas, algoritma Boyer-Moore tergolong cukup mangkus dibandingkan dengan beberapa algoritma lainnya seperti *brute force* ataupun *Knuth Morris Pratt algorithm* dikarenakan perbandingan karakternya yang berasal dari belakang sehingga dapat membuat perbandingan karakter yang lebih sedikit dibandingkan algoritma lainnya.

Namun, algoritma yang penulis buat masih mengalami beberapa kendala, seperti contohnya apabila kamus kata yang memiliki prefix dan sufix yang sama sehingga program sedikit bingung memutuskan apakah kata tersebut memiliki rima kata yang sama atau tidak dengan *pattern*-nya. Selain itu beberapa algoritma seperti *metaphone* cukup baik dalam melakukan pencarian rima kata karena sudah didasari dengan pemahaman kata dari algoritmanya. Namun secara garis besar, algoritma Boyer-Moore yang diperkuat dengan *regular expression* sudah cukup baik untuk melakukan *string matching* suku kata terakhir dengan semua kata di dalam kamus kata.

Berikut adalah percobaan yang dilakukan kepada program dengan kasus yang sama dengan analisis kasus di atas. Hasil menunjukkan hal serupa bahwa dengan kamus yang terbatas, kata "senja" memiliki rima kata yang serupa dengan "kemboja" ataupun kata "sahaja".

```
PS E:\IF\sem4\Stima\makalah> & C:/Users/asus/AppData/Local/Programs/Python/Python37-32/python.exe e:/IF/sem4/Stima/makalah/syllabledoable.py
Kata yang ingin dicari: senja
Kata yang memiliki rima kata yang sama:
kemboja
sahaja
PS E:\IF\sem4\Stima\makalah> █
```

Gambar 9 Hasil Pencarian Rima Kata "Senja" (Sumber: Dokumen Pribadi)

Akan dicoba kembali menggunakan kasus lain. Digunakan kata "barang" dan akan dicari kembali kata apa saja yang memiliki rima kata yang sama dengan kata tersebut.

```
PS E:\IF\sem4\Stima\makalah> & C:/Users/asus/AppData/Local/Programs/Python/Python37-32/python.exe e:/IF/sem4/Stima/makalah/syllabledoable.py
Kata yang ingin dicari: barang
Kata yang memiliki rima kata yang sama:
larang
sekarang
PS E:\IF\sem4\Stima\makalah> █
```

Gambar 10. Hasil Pencarian Rima Kata "Barang" (Sumber: Dokumen Pribadi)

Algoritma Boyer-Moore akan bekerja sebagai berikut dengan contoh di atas.

s	e	k	a	r	a	n	g
			1				
r	a	n	g				

s	e	k	a	r	a	n	g
					2		
		r	a	n	g		

S	e	k	a	r	a	n	g
				6	5	4	3
				r	a	n	g

Program akan menyadari bahwa *pattern* ditemukan di akhir kalimat sehingga “sekarang” memiliki rima kata yang sama dengan “barang”.

V. LINK VIDEO YOUTUBE

Sebagai pendukung dari makalah yang telah penulis buat, penulis juga menyertakan *link video* Youtube untuk memudahkan pembaca dalam memahami bagaimana *regular expression* dan algoritma boyer-moore bekerja pada pencarian rima kata untuk mencari kata-kata yang memiliki rima kata dengan sebuah kata yang ingin dicari. Video diunggah di Youtube dengan link https://youtu.be/1bOG61P_f4Y.

VI. KESIMPULAN

Dari makalah ini, dapat disimpulkan bahwa pencarian kata yang memiliki rima kata yang sama dari sebuah kata akan mudah dicari dengan menggunakan konsep algoritma Boyer-Moore yang dibantu oleh *regular expression*. Algoritma Boyer-Moore sendiri adalah algoritma *string matching* yang cukup efisien dibandingkan beberapa algoritma lainnya karena pembacaan *pattern*-nya yang dari belakang *pattern*. Namun tidak dapat dipungkiri bahwa sebenarnya masih banyak lagi algoritma yang lebih mangkus untuk permasalahan *string matching* terkhusus dalam mencari rima kata.

Untuk makalah dengan tema serupa, penulis menyarankan untuk mengubah cara algoritma untuk mendapatkan kata-kata yang memiliki rima kata yang sama dikarenakan algoritma yang penulis gunakan (yaitu Boyer-Moore) masih tergolong kurang cocok untuk digunakan dalam berbagai kondisi khusus sehingga masih dibutuhkan banyaknya penyesuaian.

VII. UCAPAN TERIMAKASIH

Penulis mengucapkan terima kasih kepada pihak-pihak yang telah mendukung penulis sehingga makalah yang dibuat dapat dikerjakan dengan lancar dan baik tanpa ada hambatan apapun. Terima kasih kepada Allah SWT yang telah memberi rahmat kepada penulis sehingga penulis dapat menyelesaikan makalah ini. Terima kasih pula untuk kedua orang tua penulis atas segala kasih sayang dan dukungan yang diberikan baik secara materil maupun non-materil.

Terima kasih kepada Pak Rinaldi selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma kelas K4 atas kerja kerasnya dalam mendidik penulis sehingga pengetahuan penulis cukup untuk membuat makalah ini. Terima kasih juga kepada beliau karena sudah membuat *website* yang sangat informatif sehingga pengerjaan makalah ini menjadi sangat terbantu. Tak lupa terima kasih untuk semua referensi yang penulis gunakan selama pembuatan makalah ini.

Terima kasih terakhir untuk teman-teman teknik informatika terutama teman-teman K4 dan beberapa teman di luar kelas K4 yang telah bersedia memberi saya saran-saran untuk pengerjaan makalah ini.

REFERENSI

- [1] Khodra, Masayu Leylia. 2020. *Pencocokan String Dengan Regular Expression*, Bandung: Departemen Teknik Informatika.
- [2] Munir, Rinaldi. 2021. *Pencocokan String (String Matching)*, Bandung: Departemen Teknik Informatika.
- [3] <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf> diakses pada 8 Mei 2021 pukul 20.24 WIB
- [4] <https://titikdua.net/puisi-chairil-anwar/> diakses pada 8 Mei 2021 pukul 19.14 WIB
- [5] <https://www.geeksforgeeks.org/boyer-moore-algorithm-for-pattern-searching/> diakses pada 8 Mei 2021 pukul 20.58 WIB
- [6] <https://docs.python.org/3/library/re.html> diakses pada 10 Mei 2021 pukul 22.03 WIB

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.

Banyuwangi, 10 Mei 2021



Muhammad Dehan Al Kautsar
13519200