Aplikasi Algoritma *String Matching* padaWord Search Puzzle

Ikmal Alfaozi - 13520125
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520125@std.stei.itb.ac.id

***Abstract*—Word Seacrh Puzzle* adalah permainan untuk menemukan kata-kata dari kumpulan huruf. Permainan ini dapat diselesaikan dengan berbagai strategi seperti brute force dan backtracking. Makalah ini berisi uraian tentang algoritma *string matrching* serta penggunaannya dalam memecahkan teka-teki pencarian kata. Algoritma *string matching* yang digunakan adalah Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM). Pada makalah ini juga diberikan perbandingan algoritma KMP dan BM Dalam memecahkan teka-teki pencarian kata berdasarkan analisis program penulis untuk mencari kata dalam matriks huruf.**

Keywords—word search puzzle; string matching; kmp; booyer-moore.

I. PENDAHULUAN

Teka-teki pencarian kata adalah permainan kata di mana pemain harus menemukan beberapa kata yang sudah ditentukan pada kumpulan huruf acak yang berbentuk seperti matriks persegi panjang atau sering disebut dengan matriks karakter. Kata-kata yang dicari pada matriks karakter tersebut memiliki delapan kemungkinan arah pembacaannya, yaitu vertikal ke atas, vertikal ke bawah, horizontal ke kanan, horizontal ke kiri, diagonal kanan atas, diagonal kanan bawah, diagonal kiri atas, dan diagonal kiri bawah.

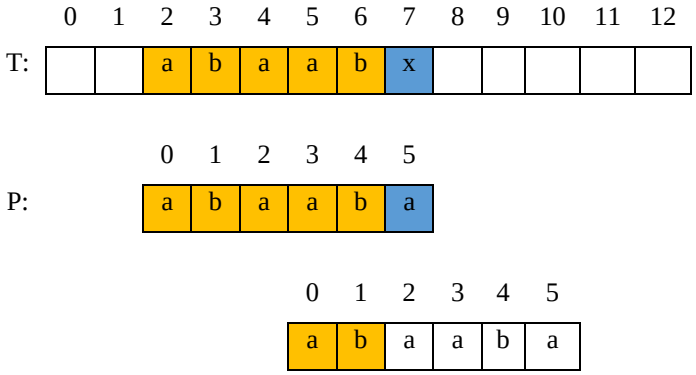
Teka-teki pencarian kata pertama kali diciptakan oleh Noeman E. Gibat. Teka-teki ini diterbitkan pada 1 Maret 1968 di Serenby Digest di Norman, Oklahoma. Kemudian, teka-teki pencarian kata menjadi populer di daerah tersebut dan beberapa guru meminta salinan untuk digunakan sebagai bahan ajar. Akhirnya, permainan menyebar setelah seorang guru mengirim salinan ke seorang teman di kota lain. Kinit teka-teki pencarian kata dapat dengan mudah kita temukan di koran, majalah bahkan di internet.

II. TEORI DASAR

A. *Knuth Morris Pratt Algorithm*

Algoritma Knuth Morris Pratt merupakan salah satu algoritma pencocokan string yang memiliki arah gerakan kiri-ke-kanan (gerakannya mirip dengan algoritma brute force). Perbedaannya terletak pada jumlah langkah switching yang dilakukan untuk mengurangi jumlah perbandingan pola yang terbuang.

Ide dari algoritma KMP ini adalah jika terjadi *mismatch* antara text dan pattern P pada indeks ke-j, misalkan T[i] ≠ P[j], maka pengecekan selanjutnya dilakukan dengan menggeser pattern sebesar prefix terpanjang dari P[0..j-1] yang sama dengan suffix dari P[1..j-1]. Berikut adalah contoh ilustrasi pergeseran pattern jika terjadi *mismatch* pada indeks ke-j.

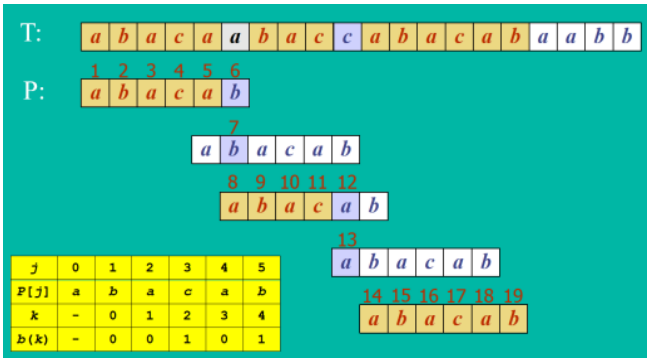


Pada contoh di atas terjadi *mismatch* pada indeks ke-5 sehingga pattern digeser sejauh panjang prefix terbesar dari “abaab” yang sama dengan suffix dari “baab”.

Knuth Morris Pratt (KMP) memiliki border function b(k) (fungsi pinggiran) yang didefinisikan sebagai ukuran terbesar dari prefix yang juga merupakan suffix pada string 0 sampi k pada pattern. Misalnya, border function dari pola “abaaba” adalah sebagai berikut:

j	0	1	2	3	4	5
P[j]	a	b	a	a	b	a
k = j - 1	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

Contoh penerapan algoritma KMP:



Gambar 1. Contoh penerapan algoritma KMP

Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

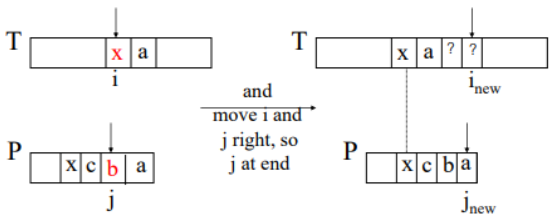
Kompleksitas dari algoritma KMP adalah O(m+n) di mana O(m) adalah kompleksitas untuk menghitung fungsi pinggiran dan O(n) adalah kompleksitas pencarian string. Kelebihan dari algoritma ini adalah tidak perlu bergerak mundur saat pencarian pada teks. Hal ini sangat bagus untuk memproses file yang sangat besar yang dibaca dari perangkat eksternal atau melalui aliran jaringan. Kekurangan algoritma KMP adalah tidak bekerja dengan baik Ketika ukuran alfabetnya meningkat sehingga mengakibatkan kemungkinan terjadinya *mistmatch* lebih banyak. *Mismatch* pada ukuran alfabet yang besar cenderung terjadi di awal pola, sedangkan KMP akan lebih cepat ketika *mismatch* terjadi mendekati akhir dari pola.

B. Boyer Moore Algorithm

Algoritma Boyer-Moore adalah salah satu algoritma pencocokan string yang paling banyak digunakan karena memiliki kompleksitas waktu yang jauh lebih baik daripada algoritma brute force. Algoritma Boyer-Moore didasarkan pada dua Teknik, yaitu *looking-glass* dan *character-jump*. Pada teknik *looking-glass*, pencocokkan pola P pada teks T dilakukan dari indeks terakhir/terbesar, tetapi pemeriksaan terhadap T tetap dimulai dari indeks terkecil.

Teknik *character jump* diterapkan saat terjadi *mismatch*. Ada 3 kondisi yang mungkin terjadi pada saat *mismatch*, yaitu sebagai berikut:

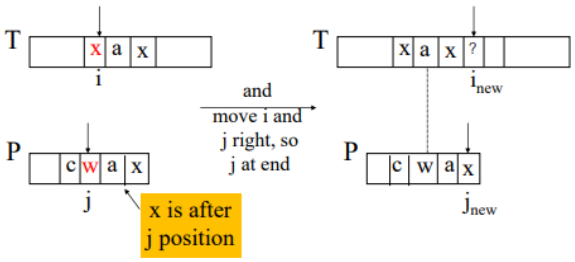
1. Terjadi *mismatch* pada T[i] dan P[j], dengan T[i] terdapat pada P, tetapi dengan indeks yang lebih kecil daripada j (i < j). Pada *mismactch* ini dilakukan penggeseran pola P ke kanan sampai *last occurrence* (kemunculan terakhir) dari T[i] pada pola P.



Gambar 2. Kasus *mismatch* 1 Boyer Moore

Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

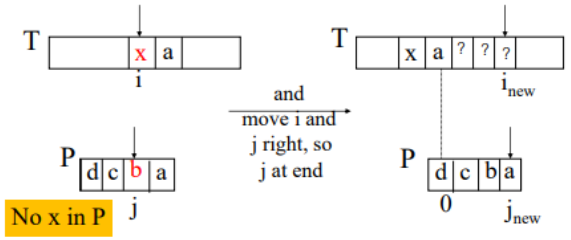
2. Terjadi *mismatch* pada T[i] dan P[j], dengan T[i] terdapat dalam pola P, tetapi dengan indeks yang lebih besar daripada j (i > j). Pada *mismatch* ini dilakukan penggeseran pola P ke kanan sebanyak satu karakter ke kanan.



Gambar 3. Kasus *mismatch* 1 Boyer Moore

Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

3. Terjadi mistmatch pada T[i] dan P[j], dengan T[i] tidak terdapat dalam pola P. Pada *mismatch* ini dilakukan penggeseran pola P ke kanan sampai indeks pertama pada pola P berada pada indeks i (indeks terjadinya *mismatch*) ditambah satu.



Gambar 4. Kasus *mismatch* 1 Boyer Moore

Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

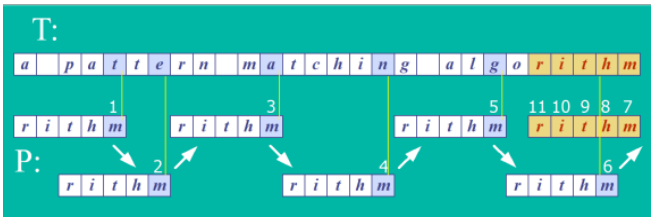
Untuk menyelesaikan kasus 1 dan 2, kita memerlukan informasi tentang indeks karakter terakhir yang muncul di T pada pola P. Oleh karena itu, diperlukan fungsi Last Occurance L(x) yang memeriksa indeks kemunculan terkahir setiap karakter pada T dalam pola P. Jika huruf T tidak muncul di P, maka nilai L(x) adalah -1. Berikut adalah contoh fungsi Last Occurance dari variasi karakter yang ada di T: X = {a, b, c, d} pada pola “abacab”.

x	a	b	c	d
L(x)	4	5	3	-1

Secara umum, prosedur untuk memeriksa kecocokan string T dan pola P adalah sebagai berikut. Pertama, hitung nilai L(x) untuk semua variasi karakter T pada pola P. Biasanya, nilai dari L(x) ini disimpan dalam bentuk tabel. Kemudian, periksa

kecocokan antara string T dan pola P, dimulai dengan indeks terkakhir dari P (*looking-glass*). Pengecekan dilanjutkan dari indeks terbesar ke indeks terkecil pada P. Jika ada *mismatch* saat melakukan pencocokkan karakter, periksa jenis *mistmatch* tersebut (kasus 1/ kasus 2/ kasus 3). Selanjutnya, geser indeks i (sebagai indeks awal dari pencocokkan karakter yang baru) sesuai dengan kasus setiap proses.

Contoh penerapan algoritma Boyer Moore:



Gambar 5. Contoh penerapan algoritma Boyer Moore

Source:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Kelebihan dari algoritma Boyer Moore adalah sangat baik ketika diaplikasikan pada pencocokkan string pada variasi karakter yang sangat beragaram, seperti karakter pada bahasa Inggris, dengan kompleksitas waktu terburuknya adalah O(mn + X), dengan X adalah jumlah karakter yang unik pada string T. Sebaliknya, kekurangan dari algoritma ini terjadi ketika variasi karakternya sangat sedikit, misal string biner.

III. ALGORITMA STRING MATCHING PADA WORD SEACRH PUZZLE

Pada word search puzzle, suatu kata dapat ditemukan dengan 8 kemungkinan arah pembacaan, yaitu vertikal ke atas, vertikal ke bawah, horizontal ke kanan, horizontal ke kiri, diagonal kanan atas, diagonal kanan bawah, diagonal kiri atas, dan diagonal kiri bawah. Oleh karena itu, pada saat pencocokkan string perlu juga diperiksa 8 arah pembacaan tersebut. Berikut adalah contoh word search puzzle serta arah pembacaannya.

Arah pembacaan horizontal

D	A	O	L	D	N	E	K	C	A	B	O	N	D	R
M	P	T	R	A	N	S	A	C	T	I	O	N	O	E
N	M	E	N	O	I	T	U	B	I	R	T	S	I	D
R	S	O	R	I	S	K	T	S	O	C	T	E	L	E
U	T	U	N	F	E	C	I	R	P	O	S	C	O	M
T	N	P	E	E	O	H	S	A	C	R	E	U	F	P
E	E	T	A	X	Y	R	U	K	C	A	R	R	T	T
R	M	A	R	C	P	M	M	H	A	T	I	I	R	I
E	E	A	R	L	O	E	A	A	F	E	E	T	O	O
M	T	S	S	N	Q	R	N	R	N	S	S	I	P	N
O	A	S	P	U	G	U	C	S	K	C	S	E	L	L
C	T	E	I	E	F	U	N	I	E	E	E	S	N	D
N	S	T	S	E	V	N	I	I	M	S	T	E	R	M
I	Y	S	S	E	V	I	T	I	T	E	P	M	O	C
S	M	A	L	L	C	A	P	A	C	E	G	R	A	L

Arah pembacaan vertikal

D	A	O	L	D	N	E	K	C	A	B	O	N	D	R
M	P	T	R	A	N	S	A	C	T	I	O	N	O	E
N	M	E	N	O	I	T	U	B	I	R	T	S	I	D
R	S	O	R	I	S	K	T	S	O	C	T	E	L	E
U	T	U	N	F	E	C	I	R	P	O	S	C	O	M
T	N	P	E	E	O	H	S	A	C	R	E	U	F	P
E	E	T	A	X	Y	R	U	K	C	A	R	R	T	T
R	M	A	R	C	P	M	M	H	A	T	I	I	R	I
E	E	A	R	L	O	E	A	A	F	E	E	T	O	O
M	T	S	S	N	Q	R	N	R	N	S	S	I	P	N
O	A	S	P	U	G	U	C	S	K	C	S	E	L	L
C	T	E	I	E	F	U	N	I	E	E	E	S	N	D
N	S	T	S	E	V	N	I	I	M	S	T	E	R	M
I	Y	S	S	E	V	I	T	I	T	E	P	M	O	C
S	M	A	L	L	C	A	P	A	C	E	G	R	A	L

Arah pembacaan diagonal

D	A	O	L	D	N	E	K	C	A	B	O	N	D	R
M	P	T	R	A	N	S	A	C	T	I	O	N	O	E
N	M	E	N	O	I	T	U	B	I	R	T	S	I	D
R	S	O	R	I	S	K	T	S	O	C	T	E	L	E
U	T	U	N	F	E	C	I	R	P	O	S	C	O	M
T	N	P	E	E	O	H	S	A	C	R	E	U	F	P
E	E	T	A	X	Y	R	U	K	C	A	R	R	T	T
R	M	A	R	C	P	M	M	H	A	T	I	I	R	I
E	E	A	R	L	O	E	A	A	F	E	E	T	O	O
M	T	S	S	N	Q	R	N	R	N	S	S	I	P	N
O	A	S	P	U	G	U	C	S	K	C	S	E	L	L
C	T	E	I	E	F	U	N	I	E	E	E	S	N	D
N	S	T	S	E	V	N	I	I	M	S	T	E	R	M
I	Y	S	S	E	V	I	T	I	T	E	P	M	O	C
S	M	A	L	L	C	A	P	A	C	E	G	R	A	L

Prosedur pencarian kata di dalam matriks karakter dengan algoritma string matching (KMP dan Boyer Moore):

1. Baca masukan file yang berisi matriks karakter, yang dipisahkan dengan spasi, dan daftar kata yang akan dicari. Simpan matriks karakter tersebut ke dalam array dua dimensi dan simpan juga kata yang akan dicari

pada matriks tersebut ke dalam sebuah array string. Berikut adalah contoh isi dari masukan file:

R M A R C P M M H A T I I R I
E E A R L O E A A F E E T O O
M T S S N Q R N R N S S I P N
O A S P U G U C S K C S E L L
C T E I E F U N I E E E S N D
N S T S E V N I I M S T E R M
I Y S S E V I T I T E P M O C
S M A L L C A P A C E G R A L

ASSETS
BACKEND
LOAD
BOND
CASH
CHARGES
COMPETITIVE
COST
DISTRIBUTION
EARN

2. Simpan text dengan arah diagonal ke dalam array string. Hal ini bertujuan agar tidak terjadi pembacaan text dengan arah diagonal yang berulang-ulang pada setiap pencarian kata pada matriks karakter.
3. Lakukan pencocokan setiap kata dengan matriks karakter. Pencocokan ini bisa dimulai dari teks dengan arah horizontal, arah vertikal, lalu diagonal. Pencocokan dilakukan dengan menggunakan algoritma KMP atau Boyer Moore.
4. Tampilkan hasil dari pencarian kata pada matriks karakter tersebut dalam bentuk matriks yang diberi penanda kata yang ditemukan pada matriks karakter tersebut, misal warna.

IV. IMPLEMENTASI

Pada makalah ini, penulis menggunakan bahasa pemrograman Python untuk implementasi *word search puzzle* dengan algoritma *string matching*. Alasan penulis menggunakan bahasa Python adalah mudah untuk dimengerti dan mempunyai *library* yang memudahkan dalam mengelola matriks dan menampilkan hasil matriks pencarian kata dengan warna. Penulis memanfaatkan *library* numpy untuk pemrosesan matriks dan colorma untuk menampilkan matriks hasil pencarian kata dengan warna. Berikut adalah penjelasan fungsi dan prosedur serta algoritma utama yang diimplementasikan dalam bahasa Python.

A. *Algoritma KMP*

Algoritma ini diimplementasikan menjadi dua fungsi, yaitu *kmpMatch* dan *computeFail*. Fungsi *computeFail* menerima masukan pattern dan mengembalikan *border function* dari pattern tersebut. Fungsi *kmpMatch* menerima masukan text dan pattern, dan mengembalikan pattern indeks awal ditemukannya pattern di dalam text tersebut. Jika pattern tidak ditemukan, fungsi *kmpMatch* akan mengembalikan -1.

```
# File KMP.py

def kmpMatch(text, pattern):
    n = len(text)
    m = len(pattern)

    fail = computeFail(pattern)

    i = 0
    j = 0

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

def computeFail(pattern):
    fail = [0 for i in range(len(pattern))]

    m =  len(pattern)
    j = 0
    i = 1

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

B. Algoritma Boyer Moore

Algoritma ini diimplementasikan menjadi dua fungsi, yaitu *bmMatch* dan *buildLast*. Fungsi *buildLast* menerima masukan pattern dan mengembalikan *last occurrence* dari semua karakter alphabet pada pattern. Fungsi *bmMatch* menerima masukan text dan pattern, dan mengembalikan indeks awal ditemukannya pattern pada teks tersebut. Jika pattern tidak ditemukan, fungsi *bmMatch* akan mengembalikan -1.

```
# File BM.py

def bmMatch(text, pattern):
    last = buildLast(pattern)
    n = len(text)
    m = len(pattern)
    i = m-1

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

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

def buildLast(pattern):
    last = [-1 for i in range(128)]

    for i in range(len(pattern)):
        last[ord(pattern[i])] = i

    return last
```

C. Program Utama

Pada *main program* terdapat fungsi reverse dan prosedur main. Fungsi reverse menerima masukan array karakter dan mengembalikan array karakter tersebut dengan urutan yang terbalik. Prosedur main merupakan program yang menjalankan langkah-langkah dalam menyelesaikan permaianan *word search puzzle*. Karena kode program terlalu panjang, penulis hanya menampilkan bagian saja. Untuk source kode lengkap dapat dilihar pada link berikut ini https://github.com/ikmalalfaozi/Word-Search-Puzzle-String-Matching.

```
# File main.py

import KMP
import BM
import colorama
from colorama import Fore
import numpy as np

def main():
…

def reverse(array):
    li = []
    for i in range(len(array) - 1, -1, -1):
        li.append(array[i])
    return li

if __name__ == "__main__":
    main()
```

Pada prosedur *main* di program utama ini terdapat pemanggilan fungsi *kmpMatch*. Jika, ingin menggunakan fungsi *bmMoore*, maka kita bisa mengganti fungsi *kmpMatch* tersebut dengan fungsi *bmMoore*.

V. HASIL

Percobaan 1

Masukan isi file:

D A O L D N E K C A B O N D R
M P T R A N S A C T I O N O E
N M E N O I T U B I R T S I D
R S O R I S K T S O C T E L E
U T U N F E C I R P O S C O M
T N P E E O H S A C R E U F P
E E T A X Y R U K C A R R T T

R M A R C P M M H A T I I R I
E E A R L O E A A F E E T O O
M T S S N Q R N R N S S I P N
O A S P U G U C S K C S E L L
C T E I E F U N I E E E S N D
N S T S E V N I I M S T E R M
I Y S S E V I T I T E P M O C
S M A L L C A P A C E G R A L

ASSETS
BACKEND
LOAD
BOND
CASH
CHARGES
COMPETITIVE
COST
DISTRIBUTION
EARN
EQUITY
EXPENSES
FEE
INCOME
INVEST
MONEY
MARKET
PERFORMANCE
PORTFOLIO
PRICE
RATES
REDEMPTION
RETURN
RISK
RRSP
SECURITIES
SELL
SERIES
SMALL
STATEMENTS
STOCK
TERM
TRANSACTION
UNIT

Output Menggunakan KMP:

```
Masukkan path relative file: small.txt
D A O L D N E K C A B O N D R
M P T R A N S A C T I O N O E
N M E N O I T U B I R T S I D
R S O R I S K T S O C T E L E
U T U N F E C I R P O S C O M
T N P E E O H S A C R E U F P
E E T A X Y R U K C A R R T T
R M A R C P M M H A T I I R I
E E A R L O E A A F E E T O O
M T S S N Q R N R N S S I P N
O A S P U G U C S K C S E L L
C T E I E F U N I E E E S N D
N S T S E V N I I M S T E R M
I Y S S E V I T I T E P M O C
S M A L L C A P A C E G R A L

--- 0.03503775596618652 seconds ---
```

Gambar 6. Output program 1

Output menggunakan Boyer Moore:

```
Masukkan path relative file: small.txt
D A O L D N E K C A B O N D R
M P T R A N S A C T I O N O E
N M E N O I T U B I R T S I D
R S O R I S K T S O C T E L E
U T U N F E C I R P O S C O M
T N P E E O H S A C R E U F P
E E T A X Y R U K C A R R T T
R M A R C P M M H A T I I R I
E E A R L O E A A F E E T O O
M T S S N Q R N R N S S I P N
O A S P U G U C S K C S E L L
C T E I E F U N I E E E S N D
N S T S E V N I I M S T E R M
I Y S S E V I T I T E P M O C
S M A L L C A P A C E G R A L

--- 0.05870699882507324 seconds ---
```

Gambar 7. Output program 2

Percobaan 2

Masukan isi file:

M M O S I R I S M E M P H I S
M O D G N I K E L D D I M S C
I D D N E F E R T I T I P I R
C G A G P A P Y R U S H V O H
R N R S N S H A B T I I M I M
O I T I S I E L I N L A E U P
L K A S I A K S X I N R M Y T
I D P T B N O W Z P O M R E P
T L O R U A S A E G Y A M H B
H O E U N L T R L N M P A S A
O C L M A I I Y K I L R I B R
R E C V O O P U D E A N T M A
U E E N D H S S S O B E K O C
S R G Y S H P C H A R I O T S
Y M U M M I F I C A T I O N T

ANUBIS
CHARIOTS
CIVILIZATION
CLEOPATRA
HIEROGLYPHS
HORUS
KUSH
MEMPHIS
MICROLITH
MIDDLE KINGDOM
MUMMIFICATION
MUMMY
NAOS
NEFERTITI
NEW KINGDOM
NILE
OLD KINGDOM
OSIRIS
PAPYRUS
PHARAOH
PYRAMIDS
ROMAN PERIOD
SCARAB
SHABTI
SISTRUM
SLAVERY
SOBEK
SPHINX
TEMPLES
TOMBS

Output menggunakan KMP:

```
Masukkan path relative file: small2.txt
M M O S I R I S M E M P H I S
M O D G N I K E L D D I M S C
I D D N E F E R T I T I P I R
C G A G P A P Y R U S H V O H
R N R S N S H A B T I I M I M
O I T I S I E L I N L A E U P
L K A S I A K S X I N R M Y T
I D P T B N O W Z P O M R E P
T L O R U A S A E G Y A M H B
H O E U N L T R L N M P A S A
O C L M A I I Y K I L R I B R
R E C V O O P U D E A N T M A
U E E N D H S S S O B E K O C
S R G Y S H P C H A R I O T S
Y M U M M I F I C A T I O N T

--- 0.0400395393371582 seconds ---
```

Gambar 8. Output program 3

Output menggunakan Boyer Moore:

```
Masukkan path relative file: small2.txt
M M O S I R I S M E M P H I S
M O D G N I K E L D D I M S C
I D D N E F E R T I T I P I R
C G A G P A P Y R U S H V O H
R N R S N S H A B T I I M I M
O I T I S I E L I N L A E U P
L K A S I A K S X I N R M Y T
I D P T B N O W Z P O M R E P
T L O R U A S A E G Y A M H B
H O E U N L T R L N M P A S A
O C L M A I I Y K I L R I B R
R E C V O O P U D E A N T M A
U E E N D H S S S O B E K O C
S R G Y S H P C H A R I O T S
Y M U M M I F I C A T I O N T

--- 0.052989959716796875 seconds ---
```

Gambar 9. Output program 4

VI. KESIMPULAN

Waktu eksekusi program untuk melakukan pencarian kata pada matriks karakter dengan algoritma KMP dan Boyer Moore sangat cepat. Dibandingkan dengan Boyer Moore, KMP terlihat memiliki waktu eksekusi program yang sedikit lebih cepat. Hal itu kemungkinan disebabkan karena eksekusi fungsi *buildLast* sedikit lebih lambat daripada fungsi *computeFail*.

VII. UCAPAN TERIMA KASIH

Pertama-tama penulis panjatkan puji syukur kehadirat Allah SWT karena atas rahmat-Nya penulis dapat menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada keluarga yang selalu mendukung penulis selama proses penulisan. Akhir kata, penulis mengucapkan terima kasih kepada semua dosen pengampu mata kuliah Strategi Algoritma, terutama Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc selaku dosen K02 atas ilmu yang telah diberikan kepada penulis sehingga penulis dapat mennyelesaikan makalah ini.

PRANALA VIDEO YOUTUBE

Berikut merupakan pranala video Youtube terkait penjelasan beserta demo makalah ini https://youtu.be/oL4XGfNJHeE.

REFERENSI

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 20 Mei 2022.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf diakses pada 20 Mei 2022.

[3] http://www-sr.informatik.uni-tuebingen.de/~buehler/BM/BM1.html diakses pada 20 Mei 2022.

[4] http://www.swingtradesystems.com/prp/books.html diakses pada 20 Mei 2022.

[5] https://www.wordsearch365.com/tips/history-of-word-search diakses pada 20 Mei 2022.

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

Ikmal Alfaozi 13520125