Application of String Matching Algorithms to Improve Word Recognition in Scrabble

Nabila Shikoofa Muida - 13522069
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: nabilashikoofa@gmail.com

***Abstract*—**Scrabble is a popular board game where players must form words from letter tiles, aiming to score as many points as possible. Effective word recognition is crucial for gameplay strategy. This paper discusses the application of string matching algorithms to enhance word recognition in Scrabble. Specifically, we highlight two algorithms: Knuth-Morris-Pratt (KMP) and Boyer-Moore (BM).

Keywords*—board game; boyer-moore algorithm; knuth-morris-pratt algorithm; scrabble; string matching*

I. Pendahuluan

Scrabble adalah sebuah *board game* yang dapat dimainkan oleh 2 hingga 4 orang dengan durasi permainan sekitar 50 hingga 90 menit. Papan permainan ini memiliki bentuk grid 15x15 di mana setiap kotak diisi dengan satu keping huruf. Setiap pemain memiliki tujuh keping huruf untuk memulai permainan. Kata yang dibuat harus berasal dari keping huruf ini, dan pemain dapat mengambil keping huruf baru setelah menggunakan yang lama. Tujuan utama permainan adalah untuk mencetak poin sebanyak-banyaknya dengan menggunakan sebanyak mungkin keping-keping huruf yang dimiliki untuk membentuk kata-kata.

Gambar 1.1. Papan Permainan Scrabble
Sumber: https://fitatsea.com/product/scrabble-original-game/

"Scrabble" berasal dari kata dalam bahasa Inggris yang berarti "berjuang membanting tulang". Permainan ini awalnya diciptakan pada tahun 1938 oleh seorang arsitek bernama Alfred Mosher Butts dengan nama "Criss-Crosswords". Scrabble merupakan penyempurnaan dari permainan Lexiko dan dimainkan dengan cara yang mirip dengan teka-teki silang.

Pemain dikatakan bermain dengan baik apabila bisa menggunakan huruf-huruf yang dimiliki untuk menyusun sebuah kata yang dapat dibaca dari kiri ke kanan, atau dari atas ke bawah. Kata utama yang disusun harus mengandung satu atau lebih huruf yang sudah diletakkan sebelumnya. Bila kata tersebut ternyata menghasilkan kata (atau kata-kata) baru yang bersambungan dengan kata yang baru disusun, poin yang tercipta dari kata tambahan ini pun ikut dihitung, dan sekaligus bisa juga dipertanyakan.

Makalah ini bertujuan untuk mengaplikasikan algoritma pencocokan string untuk meningkatkan pengenalan kata dalam permainan Scrabble. Algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) dipilih karena efisiensi mereka dalam pencocokan pola, yang diharapkan dapat mempercepat dan mempermudah verifikasi kata dalam permainan ini. Dengan menerapkan algoritma ini, diharapkan dapat memberikan manfaat berupa peningkatan kecepatan dan akurasi dalam proses verifikasi kata, yang pada akhirnya dapat meningkatkan pengalaman bermain Scrabble.

II. Landasan Teori

A. *Algoritma Knuth-Morris-Pratt (KMP)*

Algoritma Knuth-Morris-Pratt (KMP) merupakan sebuah teknik dalam pencocokan string yang mengatasi kelemahan algoritma brute force dalam hal efisiensi waktu. KMP mencari pola dalam teks dari kiri ke kanan serupa dengan pendekatan brute force tetapi melakukan ini dengan lebih cerdas. Ketika ketidakcocokan (mismatch) terjadi, KMP memindahkan pola secara efisien berdasarkan informasi yang sudah dikumpulkan sebelumnya sehingga menghindari perbandingan yang tidak perlu.

KMP menggunakan apa yang disebut sebagai "*border function*" untuk menentukan seberapa jauh pola harus digeser ketika ketidakcocokan terjadi. *Border function* menghitung dan menyimpan ukuran dari prefix terpanjang pada pola yang

juga merupakan suffix. Nilai ini disimpan dalam sebuah tabel yang digunakan untuk memutuskan seberapa jauh pola harus digeser ketika ketidakcocokan terjadi. Dengan kata lain, algoritma ini memanfaatkan informasi dari perbandingan yang telah dilakukan untuk meminimalkan jumlah perbandingan di masa mendatang.

Sebagai contoh, jika pola "ababa" memiliki ketidakcocokan setelah karakter kelima, KMP akan memeriksa tabel border dan melihat bahwa dua karakter terakhir dari string (yaitu, "ba") merupakan awalan yang juga merupakan akhiran. Oleh karena itu, pola digeser sehingga "ba" di awal pola sejajar dengan "ba" yang telah cocok di teks, sehingga perbandingan bisa dilanjutkan dari titik ini.

Dengan menggunakan *border function* ini, KMP mengurangi jumlah perbandingan yang perlu dilakukan dibandingkan dengan pencocokan pola brute force, membuatnya sangat efisien terutama untuk pencarian string yang panjang atau ketika pola yang dicari relatif panjang dan kompleks.

```
function kmpMatch(text: String, pattern: String)
-> int
    n <- length of text
    m <- length of pattern
    b[] <- computeBorder(pattern)
    i <- 0
    j <- 0

    while i < n
        if pattern[j] == text[i]
            if j == m - 1
                return i - m + 1  // match found
            i <- i + 1
            j <- j + 1
        else If j > 0
            j <- b[j - 1]
        else
            i <- i + 1
    end while

    return -1  // no match found
end function


function computeBorder(pattern: String) -> int[]
    m <- length of pattern
    b <- new array of size m
    b[0] <- 0
    j <- 0
    i <- 1

    while i < m
        if pattern[j] == pattern[i]
            b[i] <- j + 1  // j+1 chars match
            i <- i + 1
            j <- j + 1
        else if j > 0
            j <- b[j - 1]  // j follows matching
                           prefix
        else
            b[i] <- 0  // no match
            i <- i + 1
    end while

    return b
end function
```

B. Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore merupakan metode efisien untuk pencocokan pola dalam teks, terutama berkat dua teknik utama yang digunakannya: teknik *looking-glass* dan *character-jump*. Teknik *looking-glass* memulai pencarian dari akhir pola, bukan dari awal. Dengan memulai dari akhir, algoritma ini dapat mengidentifikasi ketidakcocokan lebih awal dan kemudian menggunakan informasi ini untuk melompat lebih jauh dalam teks tanpa membandingkan setiap karakter.

Ketika terjadi ketidakcocokan antara karakter di teks dan karakter di pola, algoritma ini menentukan seberapa jauh pola harus digeser berdasarkan posisi terakhir kemunculan karakter yang tidak cocok ini dalam pola. Jika karakter tidak ditemukan dalam pola, pola dapat digeser melewati posisi karakter tersebut dalam teks.

Kasus pemindahan pola yang mungkin terjadi adalah :

1. **Kasus Pertama:** Jika karakter yang tidak cocok di teks ditemukan di suatu tempat dalam pola, pola digeser sehingga kejadian terakhir karakter ini dalam pola diselaraskan dengan karakter di teks.
2. **Kasus Kedua:** Jika karakter di teks tidak ada dalam pola, pola digeser seluruhnya melewati karakter ini.
3. **Kasus Ketiga:** Jika karakter yang tidak cocok adalah unik dalam pola, maka pola digeser sejauh mungkin tanpa melewati kemungkinan kecocokan selanjutnya.

```
function bmMatch(text: String, pattern: String)
-> int
    last <- buildLast(pattern)
    n <- length of text
    m <- length of pattern
    i <- m - 1

    if i > n - 1
        return -1  // no match if pattern is
                      longer than text

    j <- m - 1

    do
        if pattern[j] == text[i]
            if j == 0
                return i  // match found
            else
                i <- i - 1
                j <- j - 1
        else
            lo <- last[text[i]]  // last
occurrence of text[i] in pattern
            i <- i + m - min(j, 1 + lo)
            j <- m - 1
```

```
    while i <= n - 1

    return -1  // no match found
end function


function buildLast(pattern: String) -> int[]
    last <- new array of size 128  //ASCII
charset

    for i from 0 to 127
        last[i] <- -1  // initialize array

    for i from 0 to length of pattern - 1
        last[pattern[i]] <- i

    return last
end function
```

C. *Langkah Penyelesaian Scrabble*

Sebelum permainan dimulai, huruf-huruf dimasukkan ke dalam kantong yang tidak tembus pandang, atau sisi keping yang berhuruf diletakkan menghadap ke bawah di atas permukaan yang rata. Selanjutnya, giliran pertama bermain ditentukan dengan mengambil sebuah keping yang terdapat di dalam kantong. Pemain yang berhasil mendapat huruf yang terdekat dengan huruf "A" mendapat giliran pertama, sedangkan keping kosong dianggap lebih superior dari huruf "A". Jika dua atau lebih pemain mengambil huruf yang sama, huruf-huruf tersebut dimasukkan kembali ke dalam kantong dan pengundian diulang dari awal.

Setiap huruf memiliki nilai tertentu (antara 1 hingga 10) yang ditentukan berdasarkan frekuensi kemunculannya dalam tulisan standar. Dalam edisi Scrabble bahasa Inggris, huruf-huruf yang sering muncul dalam kosakata bahasa Inggris, seperti "E" dan "O," hanya bernilai 1 poin. Sebaliknya, huruf-huruf yang jarang muncul, seperti "Q" dan "Z," masing-masing bernilai 10 poin. Selain itu, terdapat dua keping kosong (tidak ditulisi huruf) yang bernilai poin nol, tetapi bisa dipakai untuk melambangkan semua huruf mulai dari "A" sampai "Z".

Papan permainan memiliki kotak-kotak "bonus" yang dapat melipatgandakan jumlah poin. Kotak berwarna merah bertuliskan "triple-word" mengalikan total poin dari sebuah kata dengan 3; kotak merah muda "double-word" mengalikan total poin dari sebuah kata dengan 2; kotak biru tua "triple-letter" mengalikan nilai huruf yang ditempatkan di atasnya dengan 3; dan kotak biru muda "double-letter" mengalikan nilai huruf yang ditempatkan di atasnya dengan 2. Kotak di tengah papan permainan (H8) memiliki nilai "double-word" dan diberi tanda bintang atau logo.

Pada setiap giliran, pemain memiliki hingga 7 huruf di rak mereka yang bisa digunakan untuk membentuk sebuah kata. Saat gilirannya tiba, pemain bisa memilih salah satu dari empat tindakan berikut: (1) melewatkan giliran jika tidak bisa membentuk kata, tanpa mendapatkan poin; (2) menukar satu atau lebih huruf yang dimiliki dengan huruf dalam kantong, tanpa mendapatkan poin (ini hanya diperbolehkan jika ada minimal 7 huruf tersisa di kantong); (3) menyusun kata di papan permainan untuk mendapatkan poin; atau (4) mempertanyakan kata yang disusun oleh lawan sebelumnya dengan alasan bahwa kata tersebut tidak valid.

Setiap kali giliran selesai, pemain mengambil lagi huruf-huruf dari dalam kantong untuk mengisi rak hingga kembali berjumlah 7 buah huruf. Bila huruf yang ada di dalam kantong jumlahnya sudah kurang dari 7 buah, pemain tidak punya pilihan kecuali mengambilnya semua. Permainan berakhir apabila salah seorang pemain sudah menghabiskan semua huruf di rak, dan tidak ada lagi huruf yang tersisa di dalam kantong (walaupun di rak pemain lawan masih tersisa huruf-huruf yang belum dimainkan), atau enam kali giliran dimainkan tetapi tidak menghasilkan angka, dan paling tidak terdapat sebuah kata di atas papan permainan.

Penghitungan poin yang didapat dari kata yang dimainkan:

- Kalikan poin yang tertera pada huruf yang diletakkan di atas kotak bonus sesuai dengan petunjuk kotak bonus, kalikan 2 untuk "double-letter" atau kalikan 3 untuk "triple-letter"
- Tambahkan poin-poin yang tertera pada huruf sisanya (huruf yang baru diletakkan atau huruf yang sebelumnya sudah ada), kecuali keping kosong.
- Bila salah satu huruf yang membentuk kata diletakkan di atas kotak "double-word", maka total poin untuk kata tersebut dikalikan dua (atau dikalikan 2 lagi)
- Bila salah satu huruf yang membentuk kata diletakkan di atas kotak "triple-word", maka total poin untuk kata tersebut dikalikan tiga (atau dikalikan 3 lagi).

Bila pemain berhasil menggunakan seluruh huruf yang dimiliki (7 huruf sekaligus), maka pemain tersebut berhak atas bonus 50 poin yang ditambahkan pada total poin kata yang disusun.

III. Penerapan Algoritma

A. *Memuat Kamus*

Untuk memulai proses verifikasi kata dalam permainan Scrabble, pertama-tama kita perlu memuat kamus kata yang akan digunakan. Berikut ini adalah fungsi yang digunakan untuk membaca daftar kata dari file dictionary.txt:

```python
def load_dictionary(file_path):
    with open(file_path, 'r') as file:
        return [line.strip().upper() for line in file]
```

Gambar 3.1. Fungsi Memuat Kamus dalam Scrabble
Sumber: Dokumentasi Pribadi

Program ini dibuat menggunakan gabungan kamus dengan jumlah kata sebanyak 178691 kata.

Scrabble > dictionary.txt

```
178677  ZYMOLYSES
178678  ZYMOLYSIS
178679  ZYMOLYTIC
178680  ZYMOMETER
178681  ZYMOMETERS
178682  ZYMOSAN
178683  ZYMOSANS
178684  ZYMOSES
178685  ZYMOSIS
178686  ZYMOTIC
178687  ZYMURGIES
178688  ZYMURGY
178689  ZYZZYVA
178690  ZYZZYVAS
178691  ZZZ
```

Gambar 3.2. Cuplikan File Dictionary.txt
Sumber: Dokumentasi Pribadi

B. *Implementasi Algoritma Knuth-Morris-Pratt (KMP)*

Algoritma Knuth-Morris-Pratt menggunakan tabel prefix untuk menghindari perbandingan yang tidak perlu. Implementasi algoritma Knuth-Morris-Pratt (KMP) untuk verifikasi kata dalam Scrabble melibatkan dua fungsi utama, yaitu **`build_kmp_table`** untuk membangun tabel KMP dan **`kmp_search`** untuk melakukan pencarian kata.

```python
def build_kmp_table(pattern):
    table = [0] * len(pattern)
    j = 0
    for i in range(1, len(pattern)):
        if pattern[i] == pattern[j]:
            j += 1
            table[i] = j
        else:
            if j > 0:
                j = table[j - 1]
                i -= 1
            else:
                table[i] = 0
    return table

def kmp_search(pattern, text):
    table = build_kmp_table(pattern)
    m = len(text)
    n = len(pattern)
    i = j = 0
    while i < m:
        if pattern[j] == text[i]:
            i += 1
            j += 1
        if j == n:
            return True
        elif i < m and pattern[j] != text[i]:
            if j != 0:
                j = table[j - 1]
            else:
                i += 1
    return False
```

Gambar 3.3. Scrabble dengan Algoritma Knuth-Morris-Pratt
Sumber: Dokumentasi Pribadi

C. *Implementasi Algoritma Boyer-Moore (BM)*

Algoritma Boyer-Moore menggunakan dua teknik utama, yaitu looking-glass dan character-jump, untuk mempercepat proses pencarian dengan melakukan pencocokan dari akhir pola dan menggeser pola lebih jauh dalam teks berdasarkan informasi yang telah dikumpulkan. Implementasi algoritma Boyer-Moore (BM) untuk verifikasi kata dalam Scrabble melibatkan dua fungsi utama, yaitu **`build_last_table`** untuk membangun tabel terakhir dan **`bm_search`** untuk melakukan pencarian kata.

```python
def build_last_table(pattern):
    last = {}
    for i in range(len(pattern)):
        last[pattern[i]] = i
    return last

def bm_search(pattern, text):
    last = build_last_table(pattern)
    m = len(pattern)
    n = len(text)
    i = m - 1
    j = m - 1
    while i < n:
        if text[i] == pattern[j]:
            if j == 0:
                return True
            else:
                i -= 1
                j -= 1
        else:
            lo = last.get(text[i], -1)
            i = i + m - min(j, 1 + lo)
            j = m - 1
    return False
```

Gambar 3.4. Scrabble dengan Algoritma Boyer-Moore
Sumber: Dokumentasi Pribadi

D. *Penyelesaian Permainan Scrabble*

Untuk memeriksa apakah sebuah kata cocok dengan pola yang diberikan, berikut adalah fungsi yang digunakan dengan bantuan *Regular Expression.*

```python
import re

def matches_pattern(pattern, word):
    if pattern.startswith('_') and pattern.endswith('_'):
        regex_pattern = f".*{pattern[1:-1]}.*"
    elif pattern.endswith('_'):
        regex_pattern = f"^{pattern[:-1]}.*"
    elif pattern.startswith('_'):
        regex_pattern = f".*{pattern[1:]}$"
    else:
        regex_pattern = pattern

    match = re.fullmatch(regex_pattern, word)
    return match is not None
```

Gambar 3.5. Pattern Matching menggunakan Regular Expression
Sumber: Dokumentasi Pribadi

Skor masing-masing huruf dalam permainan Scrabble ditentukan oleh frekuensi kemunculannya dalam tulisan standar bahasa Inggris. Berikut adalah skor yang diberikan untuk setiap huruf dan fungsi untuk menghitung skor dari sebuah kata:

```python
# Skor masing-masing huruf dalam scrabble
SCRABBLE_SCORES = {
    'A': 1, 'B': 3, 'C': 3, 'D': 2, 'E': 1, 'F': 4, 'G': 2,
    'H': 4, 'I': 1, 'J': 8, 'K': 5, 'L': 1, 'M': 3, 'N': 1,
    'O': 1, 'P': 3, 'Q': 10, 'R': 1, 'S': 1, 'T': 1, 'U': 1,
    'V': 4, 'W': 4, 'X': 8, 'Y': 4, 'Z': 10
}

# Fungsi untuk menghitung skor dari sebuah kata
def calculate_score(word):
    return sum(SCRABBLE_SCORES[letter] for letter in word)
```

Gambar 3.6. Skor tiap Huruf dan Kata
Sumber: Dokumentasi Pribadi

Berikut adalah fungsi utama untuk mencari kata yang mungkin menggunakan KMP dan BM.

```python
def find_possible_words(letters, pattern, dictionary, algorithm):
    possible_words = []
    search_algorithm = kmp_search if algorithm == 'kmp' else bm_search
    for word in dictionary:
        if can_form_word(letters + pattern.replace('_', ''), word):
            if matches_pattern(pattern, word) and search_algorithm(pattern.replace('_', ''), word):
                possible_words.append((word, calculate_score(word)))
    return sorted(possible_words, key=lambda x: x[1], reverse=True)
```

Gambar 3.7. Fungsi untuk Mencari Potensi Kata
Sumber: Dokumentasi Pribadi

IV. Hasil dan Pembahasan

Pada bab ini, hasil dari penerapan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM) dalam verifikasi kata pada permainan Scrabble akan dibahas. Hasil dari pengujian yang dilakukan menghasilkan daftar kata-kata valid dari semua kemungkinan keping huruf yang dimiliki oleh pemain, diurutkan berdasarkan poin tertinggi. Analisis hasil yang diperoleh mencakup waktu eksekusi, akurasi pencocokan, serta evaluasi kinerja kedua algoritma dalam berbagai kondisi.

A. Metodologi Pengujian

Untuk menguji kinerja algoritma KMP dan BM, kami melakukan beberapa langkah berikut:

1) Persiapan Data: Menggunakan kamus dengan jumlah kata sebanyak 178691 kata.
2) Parameter Pengujian: Menggunakan berbagai pola kata yang berbeda dalam hal panjang dan kompleksitas.
3) Pengukuran Waktu Eksekusi: Mengukur waktu yang dibutuhkan oleh masing-masing algoritma untuk melakukan pencarian kata.
4) Pengukuran Akurasi Pengenalan Kata: Menentukan akurasiberdasarkan jumlah kata yang berhasil ditemukan oleh masing-masing algoritma.

B. Pengujian

TABEL I. Pengujian I

Available Letters	MAKALAH
Pattern	_A_

```
Available Letters: MAKALAH
Pattern: _A_
Loaded 178691 words from dictionary.

==========================================================
KMP algorithm found 25 possible words in 0.5736 seconds.
==========================================================
Score 12: KAMALA
Score 11: KALAM, LAKH
Score 10: ALMAH, HALMA, HAMAL
Score  9: AMAH, HALM
Score  8: HAM
Score  6: AAH, AHA, ALMA, KA, LAMA
Score  5: AH, AMA, HA, LAM
Score  4: AM, MA
Score  3: AAL, ALA
Score  2: AA, AL, LA

==========================================================
BM algorithm found 25 possible words in 0.4540 seconds.
==========================================================
Score 12: KAMALA
Score 11: KALAM, LAKH
Score 10: ALMAH, HALMA, HAMAL
Score  9: AMAH, HALM
Score  8: HAM
Score  6: AAH, AHA, ALMA, KA, LAMA
Score  5: AH, AMA, HA, LAM
Score  4: AM, MA
Score  3: AAL, ALA
Score  2: AA, AL, LA
```

TABEL II. Pengujian II

Available Letters	STIMAKU
Pattern	R_

```
Available Letters: STIMAKU
Pattern: R_
Loaded 178691 words from dictionary.

==========================================================
KMP algorithm found 26 possible words in 0.9712 seconds.
==========================================================
Score 13: RUMAKIS
Score 12: RUMAKI
Score  9: RAKIS, RAKUS
Score  8: RAKI, RAKU, RISK, RUSK
Score  7: RAMUS
Score  6: RAMI, RAMS, RIMS, RUMS
Score  5: RAM, RIM, RUM
Score  4: RAIS, RATS, RIAS, RUST, RUTS
Score  3: RAI, RAS, RAT, RIA, RUT

==========================================================
BM algorithm found 26 possible words in 0.9208 seconds.
==========================================================
Score 13: RUMAKIS
Score 12: RUMAKI
Score  9: RAKIS, RAKUS
Score  8: RAKI, RAKU, RISK, RUSK
Score  7: RAMUS
Score  6: RAMI, RAMS, RIMS, RUMS
Score  5: RAM, RIM, RUM
Score  4: RAIS, RATS, RIAS, RUST, RUTS
Score  3: RAI, RAS, RAT, RIA, RUT
```

TABEL III. PENGUJIAN III

Available Letters	ACTERIA
Pattern	_QUIT

```
Available Letters: ACTERIA
Pattern: _QUIT
Loaded 178691 words from dictionary.

==========================================================
KMP algorithm found 1 possible words in 0.9507 seconds.
==========================================================
Score 17: ACQUIT

==========================================================
BM algorithm found 1 possible words in 0.9177 seconds.
==========================================================
Score 17: ACQUIT
```

TABEL IV. PENGUJIAN IV

Available Letters	ACTERIA
Pattern	QUIT_

```
Available Letters: ACTERIA
Pattern: QUIT_
Loaded 178691 words from dictionary.

==========================================================
KMP algorithm found 2 possible words in 1.1415 seconds.
==========================================================
Score 16: QUITTER
Score 14: QUITE

==========================================================
BM algorithm found 2 possible words in 0.8548 seconds.
==========================================================
Score 16: QUITTER
Score 14: QUITE
```

TABEL V. PENGUJIAN V

Available Letters	ACTERIA
Pattern	_QUIT_

```
Available Letters: ACTERIA
Pattern: _QUIT_
Loaded 178691 words from dictionary.

==========================================================
KMP algorithm found 4 possible words in 2.9540 seconds.
==========================================================
Score 20: ACQUITTER
Score 17: ACQUIT
Score 16: QUITTER
Score 14: QUITE

==========================================================
BM algorithm found 4 possible words in 0.9485 seconds.
==========================================================
Score 20: ACQUITTER
Score 17: ACQUIT
Score 16: QUITTER
Score 14: QUITE
```

TABEL VI. HASIL PENGUJIAN WAKTU EKSEKUSI

No	Available Letters	Pattern	Waktu Eksekusi (s)	
			KMP	BM
1	MAKALAH	_A_	0.5736	0.4540
2	STIMAKU	R_	0.9712	0.9208
3	ACTERIA	_QUIT	0.9507	0.9177
4	ACTERIA	QUIT_	1.1415	0.8548
5	ACTERIA	_QUIT_	2.9540	0.9485

Dari tabel hasil pengujian waktu eksekusi, terlihat bahwa algoritma Boyer-Moore (BM) secara konsisten memiliki waktu eksekusi yang lebih cepat dibandingkan dengan algoritma Knuth-Morris-Pratt (KMP) dalam semua kasus uji. Perbedaan waktu eksekusi antara kedua algoritma lebih mencolok pada pola yang lebih kompleks, seperti pada pola "QUIT" di mana BM menyelesaikan dalam 0.9485 detik sementara KMP membutuhkan 2.9540 detik.

TABEL VII. HASIL PENGUJIAN AKURASI PENGENALAN KATA

No	Available Letters	Pattern	Kata Ditemukan	
			KMP	BM
1	MAKALAH	_A_	25	25
2	STIMAKU	R_	26	26
3	ACTERIA	_QUIT	1	1
4	ACTERIA	QUIT_	2	2
5	ACTERIA	_QUIT_	4	4

Dari tabel hasil pengujian akurasi pengenalan kata, terlihat bahwa kedua algoritma, Knuth-Morris-Pratt (KMP) dan Boyer-Moore (BM), memiliki akurasi yang sama dalam menemukan kata yang valid. Jumlah kata yang ditemukan oleh kedua algoritma adalah identik dalam semua kasus uji, menunjukkan bahwa kedua algoritma ini memiliki tingkat akurasi pengenalan kata yang setara.

Evaluasi kinerja kedua algoritma menunjukkan bahwa Boyer-Moore (BM) secara konsisten lebih cepat dalam eksekusi dibandingkan Knuth-Morris-Pratt (KMP), terutama pada pola yang lebih kompleks. Namun, perbedaan waktu yang cukup kecil ini memungkinkan KMP dalam beberapa percobaan dapat lebih cepat dibandingkan BM dengan selisih waktu yang sangat tipis. Ini menunjukkan bahwa meskipun BM lebih efisien secara umum, KMP juga memiliki potensi kinerja yang baik tergantung pada kondisi spesifik dari pola pencarian.

V. Kesimpulan

Hasil pengujian menunjukkan bahwa algoritma Boyer-Moore (BM) secara konsisten memiliki waktu eksekusi yang lebih cepat dibandingkan dengan algoritma Knuth-Morris-Pratt (KMP). Meskipun algoritma BM lebih cepat dalam eksekusi, kedua algoritma memiliki tingkat akurasi yang sama dalam mengidentifikasi kata-kata valid dari kumpulan huruf yang diberikan. Penggunaan baik algoritma KMP maupun algoritma BM diharapkan dapat meningkatkan pengalaman bermain Scrabble dengan pengenalan kata yang lebih cepat, sehingga pemain dapat bermain dengan lebih lancar dan efektif.

Acknowledgment

Penulis mengucapkan syukur kepada Tuhan yang Maha Esa atas kelancaran yang diberikan dalam penyusunan makalah berjudul "Application of String Matching Algorithms to Improve Word Recognition in Scrabble". Ucapan terima kasih juga disampaikan kepada Dr. Ir. Rinaldi Munir, M.T., Dr. Ir. Rila Mandala, dan Dr. Nur Ulfa Maulidevi sebagai dosen mata kuliah Strategi Algoritma IF2211. Pembelajaran yang luar biasa, arahan yang berharga, serta pengetahuan mendalam yang mereka bagikan sepanjang perjalanan akademik telah secara signifikan memperkaya pemahaman penulis tentang materi ini.

References

[1] Munir, Rinaldi. 2009. Diktat IF2211 Strategi Algoritma

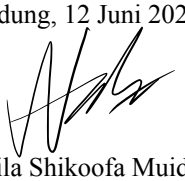
[2] Gordon, Steven A. 1994. "A Faster Scrabble Move Generation Algorithm." *Software—Practice and Experience* 24(2): 219-232.

[3] A. Schwartz, "Coding the world's fastest Scrabble program in Python," Medium, Feb. 9, 2022. [Online]. Available: https://medium.com/@aydinschwa/coding-the-worlds-fastest-scrabble-program-in-python-2aa09db670e3. [Accessed: June 12, 2024].

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, 12 Juni 2024



Nabila Shikoofa Muida
13522069