Aplikasi Algoritma *Brute Force dan String Matching* dalam Penyelesaian Permainan Kata Scrabble

Syarifa Dwi Purnamasari - 13521018
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521018@std.stei.itb.ac.id

***Abstract*—Permainan kata Scrabble merupakan salah satu permainan papan yang dimainkan secara bergiliran dan memiliki objektif untuk mencetak sebanyak mungkin poin dengan membentuk suatu kata pada papan *grid* dengan huruf-huruf yang tersedia. Untuk mendapatkan kata yang valid dan memaksimalkan poin, algoritma brute force dan string matching dapat digunakan dalam penyelesaian permainan ini.**

Keywords—brute force; boyer-moore; knuth-morris-prath; scrabble; string matching;

I. PENDAHULUAN

Permainan kata Scrabble merupakan salah satu permainan papan dengan jumlah pemain 2 sampai 4 yang memiliki objektif untuk mencetak sebanyak mungkin poin dengan membentuk suatu kata pada papan permainan yang berbentuk 15x15 *grid* dengan huruf-huruf yang tersedia. Huruf-huruf yang tersedia memiliki bentuk berupa kepingan bujur sangkar dan dapat disusun pada *grid* papan permainan sehingga membentuk sebuah kata yang valid.

Scrabble pertama kali dikenalkan pada tahun 1938 oleh Alfred Mosher Butts dengan nama "Criss-Crosswords". Permainan ini merupakan pengembangan lanjutan dari permainan sebelumnya yang telah ia ciptakan bernama Lexiko. Scrabble menggunakan keping huruf seperti Lexiko dan menggabungkannya dengan konsep permainan teka-teki silang. Alfred Butts memperhitungkan frekuensi penggunaan huruf dalam berbagai tulisan berbahasa Inggris, termasuk artikel surat kabar The New York Times, untuk menentukan distribusi huruf-huruf tersebut dalam permainan.

Pada permainan ini, setiap pemain dibekali 7 keping huruf yang diambil sendiri secara acak dari kantong. Pemain pertama memulai permainan dengan menyusun kata pada kotak yang berada di tengah papan permainan. Kata tersebut bisa disusun secara mendatar atau menurun. Setelah pemain pertama selesai, pemain berikutnya bisa menyusun kata yang berpotongan dengan huruf-huruf yang sudah disusun oleh pemain sebelumnya. Kata-kata yang terbentuk haruslah kata yang valid. Setelah menempatkan keping huruf di papan, pemain harus mengambil keping huruf baru dari kantong huruf untuk mengganti keping huruf yang telah digunakan. Setiap kali seorang pemain menyelesaikan gilirannya, mereka harus menghitung jumlah poin yang mereka peroleh. Setiap huruf memiliki nilai poin tertentu dan nilai poin total dari kata yang terbentuk akan ditambahkan ke skor pemain. Keping huruf yang sudah ditempatkan di papan *grid* tidak diperbolehkan untuk dipindahkan. Permainan berakhir ketika salah satu dari tiga kondisi terpenuhi: semua keping huruf telah digunakan, seorang pemain telah menggunakan semua keping huruf yang dimilikinya, atau tidak ada ruang kosong yang tersedia di papan untuk membentuk kata baru.

Gambar 1. Permainan Srabble
Sumber: https://cdn.britannica.com/43/242443-050-59522B48/Playing-scrabble-board-game.jpg

Pada makalah ini, penulis akan membahas program dan algoritma yang dapat digunakan untuk mendapatkan kata yang valid dan memaksimalkan poin dalam permainan Scrabble. Algoritma yang akan digunakan pada solusi penyelesaian kali ini ialah *brute force* dan *string matching*.

II. LANDASAN TEORI

A. *Algoritma Brute Force*

Algoritma *brute force* merupakan salah satu metode untuk menemukan solusi yang dapat dibilang cukup praktis dan sederhana. Dengan menggunakan algoritma *brute force*, solusi dari permasalahan tersebut pasti ditemukan walaupun dengan

waktu eksekusi yang cukup lama. Algoritma *brute force* sering digunakan karena metode pemecahan masalah ini dapat diterapkan hampir pada seluruh permasalahan.

Cara kerja dari algoritma *brute force* adalah dengan mencoba semua kemungkinan yang mungkin terjadi secara berurutan untuk mengidentifikasi dari seluruh kemungkinan tersebut, kemungkinan mana saja yang memenuhi untuk menjadi solusi permasalahan. Hal tersebut menjadikan algoritma *brute force* membutuhkan waktu eksekusi yang cukup lama. Maka dari itu, algoritma *brute force* normalnya digunakan saat input yang dimasukkan pengguna tidak terlalu besar dan pengguna tidak terlalu mengutamakan waktu eksekusi.

```
procedure SelectionSort(input/output s1, s2, ..., sn : integer)
{ Mengurutkan s1, s2, ..., sn sehingga tersusun menaik dengan metode pengurutan seleksi.
  Masukan: s1, s2, ..., sn
  Luaran: s1, s2, ..., sn (terurut menaik)  }
Deklarasi
  i, j, imin, temp : integer
Algoritma:
  for i ← 1 to n – 1  do   { jumlah pass sebanyak n – 1 }
    { cari elemen terkecil di dalam s[i], s[i+1], ..., s[n] }
    imin ← i  { elemen ke-i diasumsikan sebagai elemen  terkecil sementara }
    for j ← i+1 to n do
      if s[j] < s[imin] then
        imin ← j
      endif
    endfor
    {pertukarkan s[imin] dengan s[i] }
    temp ← s[i]
    s[i] ← s[imin]
    s[imin] ← temp
  endfor
```

Gambar 2.1. Contoh Penggunaan *Brute Force* dalam *Selection Sort*
Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf

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

Algoritma *Knuth-Morris-Pratt* (KMP) adalah algoritma pencocokan pola (pattern matching) dalam string yang dikembangkan oleh Donald Knuth, Vaughan Pratt, dan James Morris. Algoritma ini digunakan untuk mencari keberadaan sebuah string pola di dalam sebuah teks atau string utama.

Cara kerja algoritma KMP adalah dengan menghindari pencocokan ulang yang tidak perlu dari pola ke dalam teks. Hal ini dicapai dengan memanfaatkan informasi tentang pola yang telah dicocokkan sebelumnya dengan teks. Informasi ini dihasilkan dari pembangunan tabel preproses (preprocessing table) yang disebut dengan tabel π (pi).

Sebelum menggunakan algoritma KMP, perlu dilakukan inisialisasi untuk mengkomputasi border. *Compute border* ini merupakan hasil dari praproses terhadap pola yang akan diuji. Dalam konteks ini, kita memiliki pola uji yang disimbolkan sebagai P. Ketika terjadi kesalahan pencocokan (*mismatch*) pada posisi j, posisi sebelum *mismatch* tersebut disimbolkan sebagai k. *Compute border* b(k) didefinisikan sebagai ukuran terbesar dari prefiks P[0..k] yang juga merupakan sufiks dari P[1..k]. Dengan kata lain, b(k) memberikan informasi tentang panjang maksimal pola yang berupa prefiks yang juga merupakan sufiks dari bagian pola P yang telah diuji sejauh ini.

```
function compute_border(input pat: string) →
array of integer

KAMUS
    border : array [0..length of pola -
1] of integer
    I, j : integer

ALGORITHM
    border[0] ← 0
    j ← 0
    i ← 1

    while (i < length of pat) do
        if pat[j] equals pat[i] then
            border[i] ← j + 1
            j ← j + 1
            i ← i + 1
        else:
            if j is not equal to 0 then
                j ← border[j - 1]
            else:
                border[i] ← 0
                i ← i + 1

    → border
```

Gambar 2.2. Pseudocode *compute_border*
Sumber: dokumen pribadi penulis

Setelah mendapatkan border dari sebuah pola, kita dapat menggunakannya dalam algoritma utama KMP. Algoritma KMP secara formal didefinisikan sebagai berikut: Misalkan kita ingin mencari pola P dalam sebuah teks T.

```
function kmp_match(input text, pat: string) →
integer

KAMUS
    compute_border : function (input :
string) → array of integer
    border : array [0..m-1] of integer
    i, j, m, n : integer

ALGORITHM
    m ← len(pat)
    n ← len(text)
    j ← 0
    i ← 1
    border ← compute_border(pat)

    while (i < n) do
        if pat[j] equals text[i] then
            if (j=m-1) then
               -> i-m+1
            i ← i+1
            j ← j+1
         else if (j > 0) then
            j ← border[j-1]
         else
            i ← i+1
    → -1
```

Gambar 2.3. Pseudocode *kmp_match*
Sumber: dokumen pribadi penulis

Kita memiliki iterator i untuk menandai posisi iterasi pada teks, border function b yang telah diinisialisasi sebelumnya, dan iterator j untuk penelusuran pada pola P. Jika terjadi ketidakcocokan antara T[i] dan P[j], penelusuran pada iterator j akan diulang dari b[j].

C. *Algoritma Boyer-Moore (BM)*

Algoritma *Boyer-Moore* (BM) adalah algoritma pencocokan pola dalam string yang dikembangkan oleh Robert Boyer dan J Strother Moore pada tahun 1977. Algoritma ini digunakan untuk mencari keberadaan sebuah string pola dalam sebuah teks atau string utama.

Cara kerja algoritma BM adalah dengan mencocokkan pola dari belakang ke depan. Algoritma ini juga menggunakan dua heuristik. Pada heuristik karakter, jika ada karakter yang tidak cocok pada posisi tertentu, teks dapat digeser sejauh beberapa karakter ke kanan. Pada heuristik penjajaran, jika ada beberapa karakter cocok, namun ada beberapa karakter yang tidak cocok, pola dapat digeser sejauh beberapa karakter ke kiri.

Algoritma BM juga melakukan praproses pada pola yang ingin dicari dengan menggunakan sebuah fungsi pembantu yang disebut Last Occurrence Function (LOF). LOF memetakan setiap karakter A yang terdapat dalam pola P ke sebuah bilangan bulat. LOF didefinisikan sebagai L(x), di mana x adalah karakter dalam A, dan nilai L(x) adalah indeks terbesar i sehingga P[i] = x. Jika tidak terdapat indeks i dengan P[i] = x, maka nilai LOF untuk karakter x adalah -1. Nilai LOF yang didapatkan kemudian akan digunakan dalam algoritma utama BM.

```
function bm_match(input text, pat: string) →
integer

KAMUS
    last : array [0..127] of integer
    n, m, i, j, lo : integer

ALGORITHM
    n ← length of text
    m ← length of pat
    i ← m-1
    i traversal [0..127]
        last[i] ← -1
    i traversal [0..len(pat)-1]
        last[ord(pat[i])] ← i
    if (i > n-1) then
        → -1
    j ← m-1
    while (i <= n-1) do
       if pat[j] equals text[i] then
          if j equals 0 then
             → i
          else
             i ← i-1
             j ← j-1
       else
          lo ← last[text[i]]
          i ← i+m-min(j, 1+lo)
          j ← m-1
   → false
```

Gambar 2.4. Pseudocode *bm_match*
Sumber: dokumen pribadi penulis

III. IMPLEMENTASI

Program yang dibuat penulis untuk mendapatkan solusi permasalahan pada makalah kali ini ditulis menggunakan bahasa pemrograman Python. Garis besar penulisan pemrograman algoritma *string matching* BM dan KMP dibuat sama dengan yang telah dituliskan pada pseudocode di bab sebelumnya.

Pada algoritma *brute force*, penulis menggunakan library *itertools* yang menyediakan fitur *permutation* untuk menghasillkan semua kemungkinan permutasi karakter dari string bernama *chars* dengan panjang yang sama dengan panjang *rack_letters* yang diberikan pengguna. Setiap permutasi ini kemudian akan dicek kembali menggunakan fungsi *filter_combinations* yang digunakan untuk melihat apakah kombinasi permutasi karakter sesuai dengan input *rack_letters* yang diinginkan pengguna atau tidak. Jika sesuai maka karakter tersebut baru akan di *append* ke dalam list hasil yang akan di *return* oleh fungsi.

```python
from itertools import permutations

def bruteforce_match(chars, indexChar,
rack_letters):
    combinations = []

    length = len(rack_letters)
    for perm in permutations(chars,
length):
        if filter_combinations("".join(
perm), indexChar):
            combinations.append("".join(
perm))

    # print(combinations)
    return combinations

def filter_combinations(combinations,
indexChar):
    count = 0
    for j in range(len(combinations)):
        for k in range(len(indexChar)):
            if j == indexChar[k][1] and
combinations[j] == indexChar[k][0]:
                count += 1
    return count == len(indexChar)
```

Gambar 3.1. *Source Code bruteforce_match*
Sumber: dokumen pribadi penulis

Fungsi *bruteforce_match*, *kmp_match*, dan *bm_match* akan digunakan pada fungsi *scrabble*. Fungsi *scrabble* merupakan inti dari program ini. Fungsi ini membutuhkan parameter sebagai berikut.

- *dictionary* yang berisi list semua kata valid yang bisa didapatkan dari kamus
- *available_char* yang berisi keping karakter yang didapatkan pemain secara acak
- *rack_letters* yang merupakan kombinasi karakter yang ada di papan permainan dan harus berpotongan dengan *available_char*
- *point_char* yang berisi list poin tiap *character* yang dibentuk dan digunakan untuk *sorting valid words* berdasarkan poin yang lebih tinggi
- *stringMatching* yang merupakan opsi algoritma *string matching* yang dipilih pengguna untuk menemukan solusi

Fungsi *scrabble* digunakan untuk menemukan semua *possible valid words* dengan mencari semua kombinasi permutasi yang didapatkan melalui fungsi *bruteforce_match* dan pada setiap kata pada dictionary akan dicocokan dengan hasil permutasi yang telah didapatkan sebelumnya dengan menggunakan opsi *string matching* yang telah dipilih pengguna. Pada fungsi *scrabble* juga digunakan fungsi *check_string_fit* untuk mengeluarkan boolean True ketika kata pada *dictionary fit* dengan char yang tersedia.

```python
def scrabble(dictionary, available_char, rack_letters, point_char, stringMatching):
    valid_words = []
    available_char = available_char.upper()

    indexChar = []
    for i in range(len(rack_letters)):
        if rack_letters[i] != '_' and rack_letters[i] != '/':
            indexChar.append([rack_letters[i], i])

    for element in indexChar:
        available_char += element[0]
    chars = list(available_char)

    for rack in bruteforce_match(chars, indexChar, rack_letters):
        for word in dictionary:
            word = word.upper()
            if len(word) > len(available_char):
                continue
            if stringMatching == "bm":
                valid = bm_match(word,rack)
                if ((valid != -1) and (word not in valid_words) and (check_string_fit(word, available_char))):
                    valid_words.append(word)
            if stringMatching == 'kmp':
                valid = kmp_match(word,rack)
                if ((valid != -1) and (word not in valid_words) and (check_string_fit(word, available_char))):
                    valid_words.append(word)

    valid_words = sorted(valid_words, key=lambda w: sum([p[1] for p in point_char if p[0] in w]), reverse=True)
    return valid_words
```

Gambar 3.2. *Source Code scrabble*
Sumber: dokumen pribadi penulis

```python
from algorithm.bm import bm_match
from algorithm.bruteforce import bruteforce_match
from algorithm.kmp import kmp_match

def check_string_fit(string, characters):
    char_count = {}


    # Count the occurrences of each character in the characters set

    for char in characters:
        char_count[char] = char_count.get(char, 0) + 1


    # Check if the string can be formed using the characters set

    for char in string:
        if char not in char_count or char_count[char] == 0:
            return False
        char_count[char] -= 1

    return True
```

Gambar 3.3. *Source Code check_string_fit*
Sumber: dokumen pribadi penulis

Selain beberapa algoritma dan fungsi utama, penulis juga membuat fungsi lain yang digunakan untuk membantu dalam menemukan solusi penyelesaian permainan ini. Salah satu fungsi tersebut ialah *dictionary_to_array* dengan parameter path dari file. Fungsi ini digunakan untuk men-*convert* file txt *dictionary* yang berisi kumpulan kata valid yang diambil dari kamus menjadi sebuah array list yang bisa diakses program. Di samping itu, penulis juga membuat fungsi point_to_array yang memiliki objektif kurang lebih sama dengan fungsi sebelumnya tetapi digunakan untuk mengakses file poin tiap char.

Berikut merupakan *snippet main program* yang dapat dijalankan oleh pengguna.

```python
from algorithm.scrabble import *
from db.txtToArray import *
import time

dictionary = dictionary_to_array("db/dictionary.txt")
point_char = point_to_array("db/point.txt")
available_char = input("Enter available characters: ")
rack_letters = input("Enter rack letters: ")
stringMatching = input("Enter string matching (bm/kmp): ")

start_time = time.time()
valid_words = scrabble(dictionary, available_char, rack_letters, point_char, stringMatching)
end_time = time.time()

print("Valid words:", valid_words)
print("Execution time:", end_time - start_time, "seconds")
```

Gambar 3.4. *Source Code check_string_fit*
Sumber: dokumen pribadi penulis

IV. PENGUJIAN, HASIL, DAN PEMBAHASAN

Dalam pengujian efektivitas masing-masing algoritma *string matching*, pengujian akan dilakukan menggunakan 3 buah teks percobaan. Pada *testing* kali ini, file *dictionary* dan *point* yang akan digunakan sebagai berikut.

'accent', 'achene', 'aether', 'anther', 'banter', 'bather', 'beaten', 'beater', 'berate', 'bertha', 'betcha', 'branch', 'breach', 'breath', 'breech', 'cachet', 'cancer', 'canter', 'careen', 'carnet', 'center', 'centra', 'centre', 'cerate', 'cetane', 'chance', 'chebec', 'cranch', 'cratch', 'create', 'creche', 'ecarte', 'entera', 'etcher', 'ethane', 'heater', 'hereat', 'neater', 'nectar', 'nether', 'rachet', 'rebate', 'recane', 'recant', 'recent', 'reheat', 'tanrec', 'tenace', 'tenrec', 'thecae', 'thenar', 'thence', 'trance', 'trench', 'accrete', 'batcher', 'bencher', 'beneath', 'brachet', 'breathe', 'brechan', 'catcher', 'centare', 'chancer', 'chancre', 'chanter', 'cheater', 'crenate', 'earthen', 'hearten', 'hectare', 'recheat', 'reenact', 'reteach', 'teacher', 'tranche', 'bechance', 'cabernet', 'reaccent'

Gambar 4.1. *dictionary.txt*
Sumber: dokumen pribadi penulis

['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]

Gambar 4.2. *point.txt*
Sumber: dokumen pribadi penulis

Hasil dari pengujian yang dilakukan ialah berupa *list of string* yang berisi *valid words* dari semua kombinasi permutasi keping huruf yang dimiliki pemain dan diurutkan berdasarkan poin yang paling tinggi. Selain itu, akan ditampilkan juga waktu yang diperlukan tiap algoritma untuk menyelesaikan program sebagai salah satu parameter yang dapat digunakan untuk mengukur keefektifan algoritma.

1. Pengujian Pertama

available_char	CHATREB
rack_letters	_ [semua kemungkinan kata]
stringMatching	BM

Execution time : 0.004040241 s

```
Enter available characters: CHATREB
Enter rack letters: _
Enter string matching (bm/kmp): BM
Valid words: ['BATCHER', 'BRACHET', 'BETCHA', 'BREACH', 'RACHET', 'BATHER', 'BERTHA',
 'BREATH']
Execution time: 0.004040241241455078 seconds
```

available_char	CHATREB
rack_letters	_ [semua kemungkinan kata]
stringMatching	KMP

Execution time : 0.001209259 s

```
Enter available characters: CHATREB
Enter rack letters: _
Enter string matching (bm/kmp): KMP
Valid words: ['BATCHER', 'BRACHET', 'BETCHA', 'BREACH', 'RACHET', 'BATHER', 'BERTHA',
 'BREATH']
Execution time: 0.001209259033203125 seconds
```

Tabel 4.1. Tabel Pengujian Pertama
Sumber: dokumen pribadi penulis

Pada tabel pengujian pertama, *valid words* yang didapatkan kedua algoritma menunjukkan hasil yang sama. Akan tetapi, waktu yang diperlukan KMP lebih cepat sehingga pada pengujian pertama, KMP memiliki efektivitas waktu yang lebih baik dibandingkan algoritma BM.

2. Pengujian Kedua

available_char	CHATREB
rack_letters	N [semua kemungkinan kata yang memiliki huruf N]
stringMatching	BM

Execution time : 0.001996756 s

```
Enter available characters: CHATREB
Enter rack letters: N
Enter string matching (bm/kmp): BM
Valid words: ['BRECHAN', 'BRANCH', 'CHANTER', 'TRANCHE', 'TRENCH', 'ANTHER', 'THENAR'
, 'BANTER', 'CANTER', 'CARNET', 'CENTRA', 'NECTAR', 'RECANT', 'TANREC', 'TRANCE']
Execution time: 0.0019967555599975586 seconds
```

available_char	CHATREB
rack_letters	N [semua kemungkinan kata yang memiliki huruf N]
stringMatching	KMP

Execution time : 0.001337051 s

```
Enter available characters: CHATREB
Enter rack letters: N
Enter string matching (bm/kmp): KMP
Valid words: ['BRECHAN', 'BRANCH', 'CHANTER', 'TRANCHE', 'TRENCH', 'ANTHER', 'THENAR'
, 'BANTER', 'CANTER', 'CARNET', 'CENTRA', 'NECTAR', 'RECANT', 'TANREC', 'TRANCE']
Execution time: 0.0013370513916015625 seconds
```

Tabel 4.2. Tabel Pengujian Kedua
Sumber: dokumen pribadi penulis

Pada tabel pengujian kedua, *valid words* yang didapatkan kedua algoritma menunjukkan hasil yang sama. Akan tetapi, waktu yang diperlukan KMP lebih cepat sehingga pada pengujian kedua, KMP juga memiliki efektivitas waktu yang lebih baik dibandingkan algoritma BM.

3. Pengujian Ketiga

available_char	CHATREB
rack_letters	_N_ [semua kemungkinan kata yang memiliki huruf N di tengah]
stringMatching	BM
Execution time : 0.012398243 s	

```
Enter available characters: CHATREB
Enter rack letters: _N_
Enter string matching (bm/kmp): BM
Valid words: ['BRANCH', 'TRANCHE', 'CHANTER', 'TRENCH', 'ANTHER', 'THENAR', 'TRANCE',
 'BANTER', 'CANTER', 'RECANT', 'TANREC', 'CARNET', 'CENTRA']
Execution time: 0.012398242950439453 seconds
```

available_char	CHATREB
rack_letters	_N_ [semua kemungkinan kata yang memiliki huruf N di tengah]
stringMatching	KMP
Execution time : 0.009259939 s	

```
Enter available characters: CHATREB
Enter rack letters: _N_
Enter string matching (bm/kmp): KMP
Valid words: ['BRANCH', 'TRANCHE', 'CHANTER', 'TRENCH', 'ANTHER', 'THENAR', 'TRANCE',
 'BANTER', 'CANTER', 'RECANT', 'TANREC', 'CARNET', 'CENTRA']
Execution time: 0.009259939193725586 seconds
```

Tabel 4.3. Tabel Pengujian Ketiga
Sumber: dokumen pribadi penulis

Pada tabel pengujian ketiga, *valid words* yang didapatkan kedua algoritma menunjukkan hasil yang sama. Akan tetapi, sama seperti kedua pengujian sebelumnya, waktu yang diperlukan KMP lebih cepat sehingga pada pengujian ketiga, KMP memiliki efektivitas waktu yang lebih baik dibandingkan algoritma BM.

Dari ketiga pengujian tersebut, dapat disimpulkan bahwa program tersebut memiliki hasil pengujian yang sama, baik menggunakan algoritma BM maupun KMP. Namun, eksekusi waktu yang diperlukan algoritma BM lebih lama dibandingkan algoritma KMP. Hal ini dapat terjadi dalam beberapa kasus karena pendekatan yang bebeda dalam menghindari pencocokan ulang.

Algoritma KMP menghindari pencocokan ulang dengan menggunakan tabel preproses yang disebut tabel π (pi) atau border function. Tabel ini menyimpan informasi tentang kemungkinan awal pola yang cocok dengan pola itu sendiri. Dengan menggunakan tabel ini, algoritma KMP dapat melakukan "loncatan" dalam pencocokan dan memulai pencocokan berikutnya dari posisi yang memungkinkan. Ini mengurangi jumlah perbandingan yang harus dilakukan dalam pencocokan.

Di sisi lain, algoritma Boyer-Moore menggunakan heuristik *last occurrence* dan *mismatch shift* untuk menghindari pencocokan ulang. Meskipun heuristik ini efisien dalam banyak kasus, dalam beberapa situasi tertentu, seperti ketika pola memiliki banyak karakter yang sama di akhir, langkah-langkah yang diambil oleh algoritma Boyer-Moore mungkin tidak optimal. Ini dapat mengakibatkan beberapa pencocokan ulang yang tidak perlu dilakukan, yang dapat mempengaruhi kinerja algoritma.

V. KESIMPULAN DAN SARAN

A. Kesimpulan

Dari data hasil percobaan dapat ditarik kesimpulan bahwa algoritma *brute force* serta algoritma pencocokan string KMP dan BM dapat digunakan untuk membantu dalam menyelesaikan permainan kata Scrabble. Kedua algoritma tersebut sama-sama memberikan hasil yang sama dan valid. Akan tetapi, jika dibandingkan efektivitas waktu antara kedua algoritma tersebut, algoritma KMP lebih unggul dibandingkan algoritma BM. Hal ini dapat disebabkan karena pendekatan yang bebeda dalam menghindari pencocokan ulang pada kedua algoritma ini. Meskipun pada kasus ini algoritma KMP memiliki eksekusi waktu yang lebih sedikit, tidak menutup kemungkinan pada kasus lain akan ditemukan hasil yang berbeda. Algoritma BM memiliki keunggulan dalam kasus-kasus terburuknya dan umumnya bekerja lebih baik dengan pola yang panjang. Algoritma BM juga efektif dalam menghadapi pola yang memiliki karakteristik yang serupa di bagian akhirnya.

B. Saran

Dalam makalah ini, penulis menyadari bahwa masih ada beberapa kekurangan yang dapat diperbaiki untuk meningkatkan kualitasnya. Berikut adalah beberapa saran yang dapat membuat makalah ini menjadi lebih baik:

1. Meningkatkan efektivitas dan efisiensi program: Penulis dapat mengkaji ulang program yang digunakan dalam makalah dan mencari cara untuk meningkatkan efektivitas dan efisiensi algoritma yang diimplementasikan. Hal ini dapat mencakup pengoptimalan kode, penggunaan struktur data yang lebih efisien, atau eksplorasi algoritma alternatif yang dapat memberikan kinerja yang lebih baik.
2. Menambahkan gambar dan visualisasi: Untuk memudahkan pemahaman pembaca, penulis dapat mempertimbangkan untuk menyertakan lebih banyak gambar dan visualisasi yang menggambarkan konsep dan langkah-langkah algoritma dengan jelas..

Dengan menerapkan saran-saran ini, makalah ini memiliki potensi untuk menjadi lebih baik dengan peningkatan program yang lebih efektif dan efisien, serta penambahan gambar dan visualisasi yang mempermudah pemahaman.

UCAPAN TERIMA KASIH

Pertama dan yang paling utama, rasa syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas rahmatNya, penulis dapat menyelesaikan makalah ini dengan tepat waktu. Penulis juga ingin mengucapkan terima kasih kepada seluruh tim pengajar IF2211 Strategi Algoritma Institut Teknologi Bandung, terutamanya Bapak Ir. Rila Mandala, M.Eng., Ph.D., selaku Dosen Pembimbing K03 Strategi

Algoritma karena tanpa bimbingan beliau, makalah ini tidak dapat terselesaikan dengan baik. Di samping itu, penulis juga ingin mengucapkan rasa terima kasih kepada orang tua yang selalu mendukung penulis baik secara morel maupun materiel, serta teman-teman yang selalu memberi dukungan kepada penulis sehingga penulis mampu menyelesaikan makalah ini.

REFERENSI

[1] Munir, Rinaldi. 2021. *Bahan Kuliah IF2211 Strategi Algoritma: Algoritma Brute Force (Bagian: 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2021-2022/Algoritma-Brute-Force-(2022)-Bag1.pdf. Diakses pada 22 Mei 2023.

[2] Munir, Rinaldi. 2021. *Bahan Kuliah IF2211 Strategi Algoritma: Algoritma Pencocokan String*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. Diakses pada 22 Mei 2023.

[3] Findler, Robby. *Scrabble Rules*. https://users.cs.northwestern.edu/~robby/uc-courses/22001-2008-winter/scrabble.html. Diakses pada 21 Mei 2023.

[4] https://www.britannica.com/sports/Scrabble. Diakses pada 21 Mei 2023.

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, 22 Mei 2023

Syarifa Dwi Purnamasari 13521018