Penerapan Algoritma *Knuth-Morris-Pratt* dan *Boyer-Moore* dalam Permainan *Words of Wonders*

Moh. Mirza Maulana Ikhsan 13518010
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13518010@std.stei.itb.ac.id

***Abstrak*—Algoritma pencocokan *pattern* merupakan algoritma yang digunakan untuk mencari sebuah *pattern* yang terkandung pada suatu objek. Salah satu yang paling banyak digunakan adalah untuk mencari sebuah *pattern* pada teks berupa *string*. Algoritma yang sering digunakan untuk melakukan proses pencocokan tersebut, yaitu algoritma *Knuth-Morris-Pratt* dan algoritma *Boyer-Moore*.**

Kata kunci — pattern, KMP, Boyer-Moore, mismatch

I. Pendahuluan

Permainan teka-teki silang merupakan salah satu jenis permainan asah otak yang dapat dimainkan oleh satu orang, sehingga tidak dibutuhkan sumber daya yang besar untuk dapat menyelesaikan permainan teka-teki silang, terlebih hanya mengutamakan logika dan wawasan terhadap kosakata untuk dapat menyelesaikannya. Permainan jenis teka-teki silang ini biasanya dimainkan pada waktu-waktu luang saja, terutama pada saat di perjalanan.

Words of Wonders merupakan permainan teka-teki silang dimana pemain diajak untuk meningkatkan pengetahuan atau wawasan terkait dengan kosakata bahasa dan kemampuan mengeja sembari berkeliling dunia serta mengungkap rahasia tersembunyi dari tujuh keajaiban dunia.

Permainan ini memiliki ide yang unik, yakni pemain akan diberikan sejumlah karakter berupa huruf dimana pemain harus menyusun karakter tersebut sesuai dengan jumlah kotak yang diberikan agar karakter tersebut tersusun menjadi sebuah kosakata yang terdapat pada kamus bahasa. Selain meningkatkan pengetahuan terhadap kosakata, pemain juga disajikan dengan tampilan pada latar belakang berupa keindahan berbagai tempat di dunia.

Bagi beberapa orang menyelesaikan sebuah permainan teka-teki silang merupakan hal yang menantang, karena selain menjadi hiburan tersendiri, permainan ini juga dapat meningkatkan wawasan kita terhadap kosakata baru yang mungkin belum pernah kita dengar sebelumnya. Namun bagi sebagian yang lain, menyelesaikan permainan ini menjadi sebuah beban tersendiri, karena terbatasnya wawasan akan kosakata yang dimilikinya.

Salah satu solusi alternatif untuk menyelesaikan permainan ini adalah dengan cara membuka kamus besar bahasa untuk mencari berbagai kemungkinan kata dan menerapkannya pada permainan ini. Hal ini tentu saja akan menyita waktu dan usaha yang besar. Oleh karena itu, kita dapat menerapkan algoritma *Knuth-Morris-Pratt* dan *Boyer-Moore* dalam menentukan berbagai kemungkinan solusi dari tiap karakter yang tersedia dan membandingkan waktu pencarian antara algoritma *Knuth-Morris-Pratt* dan *Boyer-Moore* menggunakan kosakata bahasa Indonesia.

II. Dasar Teori

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

Algoritma *Knuth-Morris-Pratt* (KMP) adalah algoritma pencocokan *string* dengan cara melihat *pattern* dari kiri ke kanan. Cara kerja algoritma ini hampir sama dengan algoritma *brute force* akan tetapi lebih baik karena melakukan penggeseran *pattern* yang lebih cerdas sehingga tidak mengulang pencocokan dari indeks pertama pada *string*. Algoritma KMP ini memanfaatkan fungsi pinggiran untuk menentukan berapa banyak pergerseran *pattern* harus dilakukan.

Fungsi pinggiran KMP (KMP *Border Function)* merupakan fungsi untuk menghitung jumlah pergeseran yang dilakukan oleh *pattern* jika terjadi *mismatch*. Perhitungan ini dilakukan sebelum melakukan pencocokan *pattern* pada *string.* Perhitungan nilai fungsi pinggiran ini dilakukan dengan cara mencari indeks terbesar pada prefiks *pattern* mulai dari indeks ke-0 sampai *j-1* yang juga merupakan suffiks pada *pattern* dari indeks ke-1 sampai *j-1*. Misal terdapat *pattern*

"abacab" maka *pattern* tersebut akan diolah terlebih dahulu untuk menentukan nilai fungsi pinggiran, yaitu dengan cara menentukan indeks terbesar pada prefiks yang juga merupakan suffiks dari *pattern*.

Berikut merupakan implementasi algoritma pada bahasa *python* untuk menentukan prefiks dan suffiks pada *pattern*

```
def prefix(pattern,idx):
'''
Menentukan prefix dari sebuah pattern
Return array of string berisi prefix
'''
```

```
res = []
word = ""
try:
    # Mungkin terjadi index out of bound
    for i in range(idx):
        word += pattern[i]
        res.append(word)
except IndexError:
    print("Index pattern melebihi batas")
else:
    return res
```

```
def suffix(pattern,idx):
'''
Menentukan suffix dari sebuah pattern
Return array of string berisi suffix
'''
```

```
res = []
for i in range(idx,1,-1):
    word = ""
    while i<=idx:
        word += pattern[i-1]
        i+=1
    res.append(word)
return res
```

Setelah menentukan prefiks dan suffiks pada *pattern*, maka akan dilakukan perhitungan nilai fungsi pinggiran KMP. Berikut merupakan implementasi untuk menentukan nilai fungsi pinggiran KMP terhadap *pattern*.

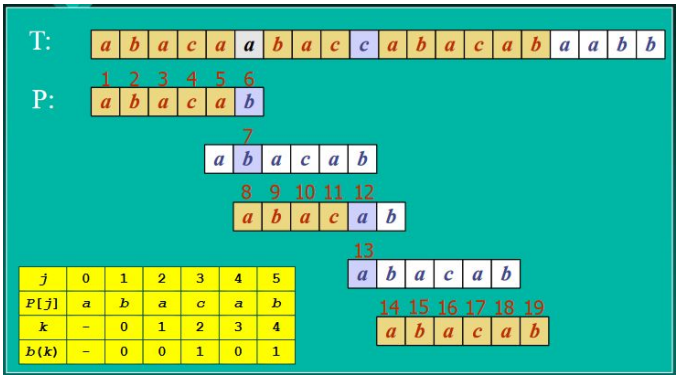
```
def idxFound(pre,suf):
'''
Return idx terbesar dari prefix yang juga
merupakan suffix dari pattern
'''
```

```
idx = 0
for i in range(len(suf)):
    if pre[i] == suf[i]:
        idx = i
if len(suf) != 0 and (pre[idx]==suf[idx]):
    return idx+1
else:
    return 0
```

```
def computeFail(pattern):
'''
Menyimpan nilai border function kedalam sebuah
array
'''
```

```
fail = [0 for i in range(len(pattern))]
for i in range(0,len(pattern)):
    fail[i] = border(pattern,i)
return fail
```

Setelah melakukan perhitungan nilai fungsi pinggiran KMP maka kita sudah dapat menerapkan algoritma KMP. Perhatikan ilustrasi *pattern matching* menggunakan algoritma KMP dengan memanfaatkan nilai fungsi pinggiran sebagai berikut:



Gambar 1 Ilustrasi cara kerja algoritma KMP untuk pencocokan *pattern*

(Sumber: Slide kuliah pencocokan *string* oleh Dr. Rinaldi Munir)

Pada contoh diatas, *pattern* yang digunakan adalah *"abacab"* dengan fungsi pinggiran KMP yang sudah dihitung. Kita coba untuk menghitung salah satu nilai fungsi pinggiran, yaitu pada $j = 4$, maka prefiksnya dari indeks ke-0 sampai *j-1* adalah *"a", "ab", "aba"*, dan *"abac"*. Sedangkan untuk suffiks dari indeks ke-1 sampai *j-1* adalah *"c", "ac"*, dan *"bac"*. Maka prefiks dari *pattern* yang juga merupakan suffiks tidak ditemukan, oleh karena itu nilai dari fungsi pinggirannya adalah nol.

Pada ilustrasi diatas dapat dilihat bahwa jika terjadi *mismatch* pada karakter *pattern* pada indeks $j = 4$ maka *pattern* akan digeser ke kanan dimulai pada indeks yang bersesuaian dengan nilai pinggiran yang sudah dihitung, yaitu nol. Hal yang sama dilakukan terus menerus hingga ditemukan *pattern* yang bersesuaian dalam *string* atau telah selesai dilakukan pengecekan hingga akhir *string*.

Berikut merupakan implementasi algoritma KMP pada bahasa *python*

```
def kmp(text,pattern):
'''
Return idx pertama pattern ditemukan pada text
atau -1 jika pattern tidak ditemukan pada text
'''
```

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

Kompleksitas waktu untuk menghitung nilai fungsi pinggiran adalah *O(m)* sedangkan untuk pencocokan *string* adalah *O(n)*, sehingga kompleksitas waktu total yang dibutuhkan untuk algoritma KMP adalah *O(m+n)*.

B. Algoritma Boyer-Moore

Algoritma *Boyer-Moore* algoritma pencocokan *string* yang didasari oleh dua teknik, yaitu teknik *the looking glass* dan teknik *the character jump*. Teknik *the looking glass* merupakan teknik pencocokan *pattern* pada *string* dimulai dari indeks terakhir *pattern* dan indeks pada *string* dimulai dari awal menyesuaikan dengan indeks *pattern*. Sedangkan teknik *the character jump* merupakan teknik melompat jika terjadi *mismatch* pada karakter *pattern* dan *string*.

Pada algoritma *Boyer-Moore* terdapat tiga kasus yang digunakan untuk teknik *the character jump*, yakni untuk kasus pertama jika pada *pattern* mengandung karakter *X* yang merupakan karakter *mismatch* pada *string,* maka geser *pattern* tersebut agar sejajar dengan karakter yang *mismatch* dengan *string*. Kasus kedua adalah jika *pattern* mengandung karakter *X* yang merupakan karakter *mismatch* pada *string* namun tidak mungkin menyejajarkan keduanya, maka *pattern* digeser sebanyak satu. Kasus ketiga terjadi ketika *pattern* tidak mengandung karakter yang terjadi *mismatch* pada *string*, maka langsung lompati *pattern* hingga melewati karakter yang menjadi *mismatch* tersebut.

Pada algoritma *Boyer-Moore* untuk menentukan seberapa banyak lompatan karakter yang dilakukan apabila terjadi *mismatch,* digunakan fungsi pendukung berupa *last occurance.* Fungsi *last occurance* ini dimiliki oleh tiap karakter berbeda pada *string*, kemudian menentukan indeks terakhir ditemukannya karakter pada *pattern*. Jika tidak ditemukan karakter *string* pada *pattern* maka nilai *last occurance* untuk karakter tersebut adalah *-1*.

Berikut merupakan implementasi algoritma *Boyer-Moore* dalam bahasa *python*.

```
def checkCharacter(text,pattern):
'''
Melakukan pengecekan pada pattern untuk
menentukan banyak karakter
berbeda
'''
```

```
dictChar = []
dictChar.append(pattern[0])
for i in range(len(pattern)):
    if pattern[i] not in dictChar:
        dictChar.append(pattern[i])
for i in range(len(text)):
    if text[i] not in dictChar:
        dictChar.append(text[i])
return dictChar
```

```
def lastOccurance(text,pattern):
'''
Return array of integer berisi posisi kemunculan
terakhir karakter dari pattern, jika char pada
pattern tidak sesuai dengan char text, maka
array akan bernilai -1
'''
```

```
char = checkCharacter(text,pattern)
posChar = [-1 for i in range(len(char))]
for i in range(len(char)):
    idx = -1
    for j in range(len(pattern)):
        if char[i] == pattern[j]:
            idx = j
    posChar[i] = idx
return posChar
```

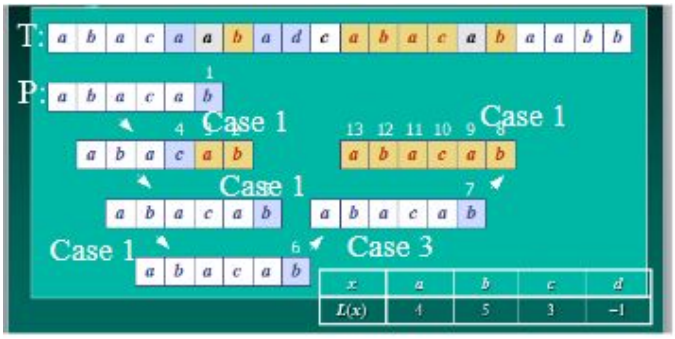
```
def bm(text,pattern):
'''
Melakukan pencocokan pattern terhadap text
Return idx ditemukannya pattern awal pada text,
-1 jika tidak ditemukan pattern pada text
'''
```

```
last = lastOccurance(text,pattern)
char = checkCharacter(text,pattern)
if len(pattern) > len(text):
    return -1
idxPatt = len(pattern)-1
idxText = len(pattern)-1
while True:
    if pattern[idxPatt] == text[idxText]:
        if idxPatt == 0:
            return idxText
        else:
            idxPatt -= 1
            idxText -= 1
    else:
              # Pengecekan terhadap kasus yang
mungkin terjadi
```

```
        idx = charAt(text[idxText],char)
        lo = last[idx]
        idxText = idxText + len(pattern) -
min(idxPatt,1+lo)
        idxPatt = len(pattern) -1
    if idxText > len(text)-1:
        break

return -1
```

Berikut merupakan ilustrasi cara kerja dari algoritma *Boyer-Moore*.



Gambar 2 Ilustrasi cara kerja algoritma *Boyer-Moore* untuk pencocokan *pattern*

(Sumber: Slide kuliah pencocokan *string* oleh Dr. Rinaldi Munir)

Pada ilustrasi diatas dapat dilihat bahwa ketika terjadi *mismatch*, maka *pattern* akan digeser ke kanan sebanyak nilai fungsi *last occurance*. Misalkan pada ilustrasi diatas, *mismatch* pertama kali terjadi pada karakter *"a"* pada *string*, sehingga *pattern* harus digeser sesuai dengan nilai dari fungsi *last occurance* sehingga karakter yang mengalami *mismatch* sejajar dengan karakter pada indeks ke-4 pada *pattern*. Hal tersebut terus dilakukan hingga ditemukannya *pattern* pada *string* atau telah selesai dilakukan pemeriksaan pada *string*.

Algoritma *Boyer-Moore* memiliki kompleksitas waktu yang hampir sama dengan KMP, namun lebih baik dibanding dengan algoritma *brute force*. Kompleksitas waktu algoritma ini adalah *O(mn + A)* dengan *A* menyatakan banyak tipe karakter yang muncul, akan tetapi pada keadaan normal kompleksitas waktu algoritma ini sama dengan algoritma KMP, yaitu *O(m+n)*.

III. Pengujian dan Analisis

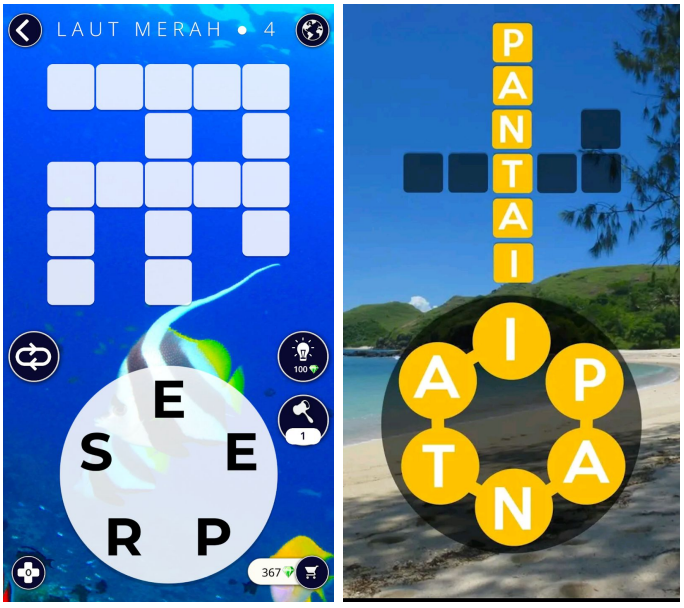
Pengujian dilakukan menggunakan program yang telah dibuat oleh penulis dengan memanfaatkan basis data kamus besar bahasa Indonesia berisi kosakata sebanyak 31.441 kata. Tujuan dari pengujian ini adalah untuk menyelesaikan permainan teka-teki silang pada *Words of Wonders* sekaligus membandingkan waktu eksekusi terhadap algoritma KMP dan *Boyer-Moore*.

A. Program Uji

Program yang dibuat oleh penulis secara umum menerapkan algoritma sebagai berikut:

1. Membaca data dari kamus besar bahasa
2. Menerima input berupa kata acak dan jumlah kotak yang disediakan berupa *integer*, kemudian mencari seluruh kombinasi karakter yang mungkin berdasarkan banyak kata sesuai masukan pengguna.
3. Melakukan pencarian pada penyimpanan dan mencocokkan tiap kata acak menggunakan algoritma KMP atau *Boyer-Moore*.
4. Jika kata terdapat pada penyimpanan, maka simpan kata tersebut.
5. Tampilkan hasil pencarian

Berikut merupakan tampilan dari permainan *Words of Wonders*.



Gambar 3 Tampilan permainan *Words of Wonders*

(Sumber: Permainan *Words of Wonders*)

Berikut merupakan tampilan *input output* program dalam melakukan pencocokan terhadap kamus besar bahasa.

```
Masukkan kata: seerp
Masukkan jumlah kata: 5
Banyak kombinasi:  60
Menghasilkan kombinasi huruf:
['peres', 'peser', 'repes', 'resep', 'serep']
5
Time execution:  0.31184983253479004
```

Gambar 4 Hasil *input output* program

```
Masukkan kata: aipant
Masukkan jumlah kata: 6
Banyak kombinasi:  360
Menghasilkan kombinasi huruf:
['pantai', 'patina', 'taipan']
3
Time execution:  1.6000714302062988
```

Gambar 5 Hasil *input output* program

B. Analisis

Analisis ini digunakan untuk membandingkan waktu eksekusi antara algoritma *Knuth-Morris-Pratt* (KMP) dan *Boyer-Moore* dalam menentukan kata yang terdapat pada kamus besar bahasa yang berisi 31.441 kosakata. Pengujian dilakukan pada laptop penulis dengan spesifikasi sebagai berikut:

- Operating System: Windows 10 Pro 64-bit
- Processor: Intel® Core™ i5-6200U CPU @ 2.30GHz (4CPUs), ~2.4GHz
- Memory: 4096 RAM

Tabel 1 Hasil pengamatan

Input kata / banyak kotak	Waktu eksekusi algoritma (detik)		Banyak kata ditemukan
	KMP	*Boyer-Moore*	
SAARP / 5	0.3647	0.3004	5
SAARP / 4	0.1531	0.1311	9
SAARP / 3	0.0467	0.0469	9
AIPANT / 6	1.5969	1.5940	3
AIPANT / 5	1.7352	1.4972	11
AIPANT / 4	0.3571	0.3517	17
REAKUL / 6	3.1777	3.2266	1
REAKUL / 5	3.0738	3.0959	11
KAGNAC / 6	1.7334	1.7194	2
KAGNAC / 5	1.7475	1.5285	6
KAGNAC / 4	0.3507	0.3206	14
TMETPA / 6	1.6486	1.5265	1
TMETPA / 5	1.7827	1.6631	7
TMETPA / 4	0.4075	0.4156	9
ELPNAT / 6	3.2518	3.2109	2
NAKALI / 6	1.7941	1.5747	1

Waktu rata-rata yang diperlukan untuk menemukan semua kemungkinan kata dalam kamus besar bahasa Indonesia untuk algoritma KMP adalah 1,4513 detik, sedangkan untuk algoritma *Boyer-Moore* waktu yang diperlukan adalah 1,3877 detik.

IV. Kesimpulan

Berdasarkan hasil pengamatan pada tabel 1, didapatkan kesimpulan bahwa algoritma *Boyer-Moore* sedikit lebih cepat dibanding dengan algoritma KMP dalam menemukan semua kemungkinan solusi yang mungkin dari kamus besar bahasa Indonesia yang berisi 31.441 kosakata. Meskipun demikian, terdapat beberapa keadaan dimana algoritma KMP lebih cepat dibanding dengan algoritma *Boyer-Moore*, yaitu ketika dalam satu kata terdapat beberapa karakter yang berulang. Algoritma *Boyer-Moore* tepat digunakan jika variasi karakter dalam suatu kata sangat variatif, sehingga akan lebih cepat dalam menemukan solusi. Meski demikian, baik algoritma KMP maupun algoritma *Boyer-Moore* waktu yang dibutuhkan masing-masing untuk menemukan solusi sudah cukup baik untuk kasus rata-rata dalam pencarian kosakata bahasa Indonesia.

Link Video Youtube

Berikut penulis lampirkan video untuk menjelaskan topik masalah pada permainan *Words of Wonders* serta implementasinya menggunakan algoritma KMP dan *Boyer-Moore*. Video dapat diakses pada https://youtu.be/Iol-354K_lM

Ucapan Terima Kasih

Rasa syukur saya ucapkan kepada Allah SWT yang telah memberikan berkat, rahmat, dan karunia-Nya sehingga saya dapat menyelesaikan pembuatan makalah ini di tengah pandemi COVID-19 saat ini. Selanjutnya ucapan terima kasih juga saya ucapkan kepada para tim dosen Strategi Algoritma yang telah memberikan saya kesempatan untuk dapat melakukan eksplorasi menambah pengetahuan. Tak lupa pula kepada dosen Dr. Masayu Leylia Khodra ST,MT yang telah mengajar di kelas K1 dengan penuh dedikasi. Harapan saya kepada makalah ini adalah semoga dapat memberikan manfaat kedepannya kepada penulis maupun para pembaca.

Referensi

[1] http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf - diakses pada 1 April 2020

[2] https://wordsofwonders.net/id/ - diakses pada 1 April 2020

[3] https://web.stanford.edu/class/cs97si/10-string-algorithms.pdf - diakses pada 2 April 2020

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Mai 2020

Moh. Mirza Maulana Ikhsan
13518010