Analisis Penerapan Algoritma Pencocokan String pada Permainan Gartic.io

Fadel Ananda Dotty 13519146
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
fadelananda123@gmail.com

***Abstract*—Salah satu algoritma yang banyak digunakan untuk memecahkan masalah dalam kehidupan sehari-hari adalah algoritma yang berkaitan dengan pencocokan string atau *string matching algorithm*. Salah satu penggunaan algoritma ini adalah untuk melakukan verifikasi jawaban dalam permainan Gartic.io. Permainan ini merupakan permainan *online multiplayer* yang cara bermainnya adalah satu orang bertugas untuk menggambar dan beberapa orang lainnya berusaha untuk menebak gambar tersebut. Algoritma pencocokan string kemudian digunakan untuk memverifikasi apakah masukan yang dimasukkan oleh penebak gambar sesuai dengan jawaban yang sebenarnya. Apabila masukan yang dimasukkan sudah benar, maka orang tersebut akan mendapatkan poin tambahan. Makalah ini akan membahas analisis terhadap berbagai algoritma pencocokan string seperti algoritma *brute force*, Knuth-Morris-Pratt, dan Boyer-Moore serta penerapan algoritma jaro similarity pada permainan Gartic.io.**

Keywords—Pencocokan String, Algoritma, Brute Force, KMP, BM, Gartic.io

I. PENDAHULUAN

Dalam dunia matematika dan ilmu komputer, terdapat banyak sekali permasalahan yang perlu dipecahkan. Salah satu cara untuk memecahkan permasalahan adalah dengan cara membuat suatu algoritma. Algoritma merupakan instruksi-instruksi yang runut dan jelas yang digunakan untuk mendapatkan suatu keluaran dari masukan dalam waktu yang terbatas. Terdapat banyak sekali algoritma yang sudah dibuat dan mereka digunakan untuk memecahkan berbagai macam permasalahan yang ada seperti persoalan pencarian, pengurutan, pencarian lintasan terpendek, dan lain-lain.

Dari berbagai macam permasalahan tersebut, salah satu permasalahan yang sering dihadapi adalah permasalahan yang berkaitan dengan pencocokan string. Permasalahan ini merupakan permasalahan pencarian tentang bagaimana cara mencari sebuah kata (*pattern*) dari teks dengan panjang tertentu yang diberikan. Banyak algoritma yang sudah dikembangkan untuk menjawab permasalahan ini. Beberapa algoritma tersebut adalah algoritma *brute force*, Boyer-Moore (BM), dan Knuth-Morris-Pratt (KMP).

Salah satu penerapan algoritma pencocokan string adalah dalam permainan Gartic.io. Permainan ini merupakan permainan *online multiplayer* yang diterbitkan oleh Gartic. Cara bermain permainan ini adalah terdapat beberapa orang yang akan masuk ke dalam ruangan yang sama, kemudian salah satu orang dari orang ini akan berperan sebagai orang yang akan menggambar dan orang sisanya berperan sebagai menebak. Penggambar ini dapat memilih satu dari dua pilihan opsi kata yang diberikan dan tugas mereka adalah menggambar kata tersebut semirip mungkin agar bisa ditebak oleh penebak. Tugas penebak adalah menebak gambar yang telah dibuat oleh penggambar dengan cara memasukkan sebuah teks ke dalam kolom yang tersedia pada *game*. Algoritma pencocokan string digunakan untuk mencocokkan apakah masukan yang telah dimasukkan oleh penebak sesuai dengan opsi kata yang telah dipilih oleh penggambar.

Gambar 1.1 Logo Permainan Gartic.io

Sumber:

https://gartic.io/download (diakses pada 7 Mei 2021)

II. LANDASAN TEORI

A. *String*

String merupakan sebuah tipe data yang digunakan untuk merepresentasikan teks pada bahasa pemrograman. String biasanya terdiri dari kumpulan karakter yang dapat berupa angka (1-9), huruf (A-Z, a-z), dan karakter spesial (&, ^, *, dll) yang dapat memiliki arti tertentu (contohnya string “ITB”) maupun tidak (contohnya string “*&(^((*&”). String biasanya digunakan untuk mengkomunikasikan informasi dari program ke penggunanya. Pada bahasa pemrograman seperti C, struktur data string biasanya direpresentasikan sebagai sebuah larik dari karakter (*array of chars*). Namun, pada bahasa pemrograman lain terdapat tipe dasar string yang dapat langsung digunakan contohnya Pyhton dan Java. Contoh deklarasi dan inisialisasi sebuah string dalam bahasa C adalah sebagai berikut:

```
char var_string[6] = {‘S’, ‘t’, ‘r’, ‘i’, ‘n’, ‘g’}
```

Pada contoh diatas, terlihat bahwa string yang dideklarasikan memiliki bentuk lengkap “String” dan disimpan

dalam *array of chars* bernama var_string. String tersebut memiliki panjang 6 dan apabila divisualisasikan, akan terdapat sebuah tabel dengan 6 buah kolom yang tiap kolomnya merepresentasikan tiap karakter pembentuk string tersebut.

Karakter	S	t	r	i	n	g
Indeks	0	1	2	3	4	5

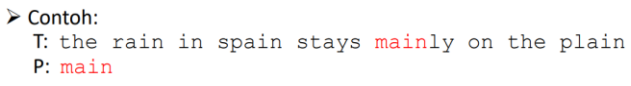
Tabel 2.1.1 Visualisasi String pada Larik

Terdapat beberapa operasi yang dapat diterapkan pada string dan operasi ini didukung oleh beberapa bahasa pemrograman tertentu. Beberapa operasi ini adalah konkatenasi (menggabungkan dua buah string yang berbeda menjadi satu), kapitalisasi (menjadikan semua karakter yang ada pada string menjadi semuanya huruf besar/huruf kecil), *split/tokenize* (menjadikan satu buah string utuh menjadi beberapa bagian berdasarkan sebuah *delimiter*), dan komparasi (membandingkan kesamaan dua buah string).

B. Algoritma Pencocokan String

Algoritma merupakan instruksi-instruksi yang runut dan jelas yang digunakan untuk mendapatkan suatu keluaran dari masukan dalam waktu yang terbatas. Biasanya algoritma digunakan untuk memecahkan permasalahan tertentu seperti persoalan pencarian, pengurutan, pencarian lintasan terpendek, dan lain-lain.

Salah satu permasalahan yang sering dihadapi dalam dunia ilmu komputer adalah permasalahan tentang pencocokan string (*string matching*). Permasalahan ini merupakan permasalahan pencarian tentang bagaimana cara mencari sebuah kata (*pattern*) dengan panjang m karakter dari teks dengan panjang tertentu (n) yang diberikan (asumsi m << n). Permasalahan ini dapat diaplikasikan menjadi berbagai macam hal seperti pencarian pada *search engine*, analisis citra, pencarian rantai kode protein pada untai DNA, dll. Berbagai algoritma yang sudah dikembangkan untuk menyelesaikan permasalahan ini yaitu algoritma *brute force*, Knuth-Morris-Pratt (KMP), dan Boyer-Moore (BM).



Gambar 2.2.1 Permasalahan Pencocokan String dengan T Sebagai Teks dan P Sebagai Pattern

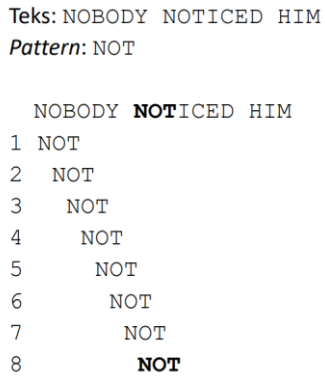
Sumber:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (diakses pada 7 Mei 2021)

Dalam melakukan pencocokan string, terdapat dua istilah yang perlu dipahami yaitu *prefix* dan *suffix*. Misalnya sebuah string S dengan panjang *m* yang dinotasikan sebagai S = $X_0X_1..X_{m-1}$. *Prefix* didefinisikan sebagai semua substring dari S yang dimulai dari indeks ke-0 sampai indeks ke k. Contohnya string "TEKNIK" memiliki *prefix* yaitu "T", "TE", "TEK", "TEKN", "TEKNI", dan "TEKNIK". *Suffix* didefinisikan sebagai semua substring S yang dimulai dari indeks ke k hingga indeks ke m-1 dimana k merupakan nilai diantara 0 sampai m-1. Contohnya string "TEKNIK" memiliki *suffix* yaitu "K", "IK", "NIK", "KNIK", "EKNIK", dan "TEKNIK". Istilah *prefix* dan *suffix* perlu untuk dipahami karena akan dipakai dalam berbagai algoritma tentang pencocokan string pada makalah ini.

C. Algoritma Brute-Force

Algoritma *brute force* merupakan salah satu pendekatan yang digunakan untuk mengatasi permasalahan pencocokan string. Algoritma ini mudah untuk diaplikasikan karena algoritma ini menggunakan pendekatan secara langsung yaitu pencarian *pattern* pada suatu teks dilakukan secara sekuensial mulai dari teks yang paling kiri dan berjalan ke kanan. Penelusuran dengan menggunakan algoritma ini dilakukan dengan cara membandingkan karakter yang ada pada *pattern* terhadap karakter pada teks satu per satu. Apabila semua karakter yang dibandingkan sama semua, maka pencarian berhasil. Apabila pada proses pencarian ditemukan karakter yang tidak sesuai, maka *pattern* akan digeser ke kanan sebanyak satu kali.



Gambar 2.3.1 Ilustrasi Pencocokan String dengan Algoritma *Brute Force*

Sumber:

https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (diakses pada 7 Mei 2021)

Algoritma *brute force* memiliki kompleksitas waktu terburuk O(mn), terbaik O(n), dan rata-rata O(m+n). Algoritma ini cenderung cepat ketika digunakan memproses teks dengan alfabet yang banyak. Namun, algoritma ini cenderung lambat ketika memproses teks dengan alfabet yang sedikit seperti *binary files*.

D. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt merupakan algoritma yang digunakan untuk melakukan pencarian string. Algoritma ini diciptakan oleh *computer scientist* Donald Knuth dan Vaughan Pratt, dan juga secara independen oleh James H. Morris pada tahun 1977 yang merupakan hasil analisis dari algoritma *brute force* yang naif.

Algoritma KMP mencocokkan *pattern* dan teks dari kiri ke kanan sama seperti algoritma *brute force*. Namun algoritma KMP ini dianggap lebih efisien daripada algoritma *brute force* karena algoritma ini melakukan proses penggeseran dengan lebih "pintar". Algoritma *brute force* hanya melakukan pergeseran sebanyak satu kali apabila terdapat kesalahan dalam pencocokan, sementara algoritma KMP dapat melakukan pergeseran lebih dari satu kali dalam kasus

tertentu. Pergeseran yang dilakukan berdasarkan *prefix* terbesar dari *pattern* P[0..j-1] yang juga merupakan *suffix* dari P[1..j-1].

Algoritma KMP melakukan *preprocessing* terhadap *pattern* yang akan dicari dari teks. *Preprocessing* ini digunakan untuk mencari *prefix* yang sesuai dari *pattern* dengan *pattern* itu sendiri. *Preprocessing* ini sering disebut dengan fungsi pinggiran atau *border function* atau juga bisa disebut *failure function* yang didefinisikan sebagai besarnya *prefix* terbesar P[0..k] yang juga merupakan *suffix* P[1..k] dengan k = j-1. Fungsi pinggiran ini perlu dieksekusi pertama kali karena dengan fungsi ini, kita dapat menghindari perbandingan yang redundan.

P: abaaba

j: 012345

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

Tabel 2.4.1 Contoh Perhitungan *Border Function* Pattern P

Sumber:

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

Pada tabel diatas, terlihat bahwa b(5) nilainya sama dengan 2. Nilai b(5) memiliki arti bahwa kita harus mencari *prefix* terbesar dari P[0..4] (“abaab”) yang juga merupakan *suffix* dari P[1..4] (“baab”). Dalam kasus ini *prefix* terbesarnya adalah “ab” yang memiliki panjang 2. Oleh karena itu nilai b(5) == 2.

Setelah menghitung fungsi pinggiran, maka hal selanjutnya adalah melakukan pencocokan string mulai dari sebelah kiri kemudian bergerak ke kanan. Fungsi pinggiran akan digunakan untuk menentukan banyaknya pergeseran apabila terjadi ketidakcocokan antara karakter pada *pattern* dan teks yang sedang dicocokkan.

Gambar 2.4.1 Ilustrasi Pencocokan String dengan Algoritma KMP

Sumber:

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

Dari ilustrasi di atas, terlihat bahwa pergeseran yang dilakukan setiap terjadi ketidakcocokan tidak hanya dilakukan sekali melainkan sesuai dengan fungsi pinggiran yang telah dihitung diawal. Hal ini menyebabkan algoritma ini lebih efektif apabila dibandingkan dengan algoritma *brute force*.

Kompleksitas waktu yang dibutuhkan untuk menghitung fungsi pinggiran adalah O(m). Selain itu, algoritma KMP sendiri membutuhkan kompleksitas waktu sebesar O(n). Oleh karena itu, secara keseluruhan algoritma KMP membutuhkan kompleksitas waktu sebesar O(m+n).

Keuntungan dari algoritma KMP adalah algoritma ini tidak perlu melakukan pergerakan secara mundur dan algoritma ini sangatlah bagus apabila digunakan untuk memproses file dengan ukuran yang besar. Kelemahan dari algoritma KMP adalah algoritma ini tidak efisien seiring dengan bertambahnya alfabet karena *mismatch* akan mungkin lebih banyak terjadi.

E. Algoritma Boyer-Moore (BM)

Algoritma Boyer-Moore merupakan salah satu algoritma yang digunakan untuk melakukan pencarian string. Algoritma ini dikembangkan oleh Bob Boyer dan J Strother Moore pada tahun 1977. Algoritma ini menggunakan dua pendekatan (heuristik) yaitu teknik *looking glass* dan teknik *character jump* yang berbeda dari algoritma *brute force* dan KMP.

Teknik *looking glass* yang dimaksud yaitu pencocokan yang dilakukan antara *pattern* P dan teks T dimulai dari belakang pattern P dan bergerak kedepan oleh algoritma ini. Teknik ini dikombinasikan dengan *character jump* agar pergeseran algoritma BM lebih efektif.

Teknik *character jump* merupakan teknik yang dilakukan untuk menggeser *pattern* apabila ditemukan ketidakcocokan pada saat melakukan pencarian dengan teknik *looking glass*. Terdapat tiga kasus tertentu pada saat melakukan *character jump*, kasus tersebut yaitu:

i. Kasus 1

Apabila terjadi ketidakcocokan antara karakter pada teks T[i] dan *pattern* P[j] dan karakter T[i] terdapat pada sebelah kiri *pattern* yang dicocokkan, maka geser P sehingga karakter tersebut sejajar dengan T[i].

ii. Kasus 2

Apabila terjadi ketidakcocokan antara karakter pada teks T[i] dan *pattern* P[j] dan karakter T[i] terdapat pada P namun pergeseran ke kanan pada karakter terakhir T[i] pada P tidak mungkin dilakukan, maka P digeser sebanyak 1 karakter ke kanan T[i+1].

iii. Kasus 3

Apabila terjadi ketidakcocokan antara karakter pada teks T[i] dan *pattern* P[j] dan karakter T[i] tidak terdapat pada *pattern*, maka P akan digeser sehingga P[0] akan sama dengan T[i+1].

Pada algoritma BM, terdapat suatu fungsi yang bernama *last occurrence function* L(). Fungsi ini dieksekusi pada awal algoritma dan digunakan untuk melakukan *preprocessing* terhadap *pattern* P. Fungsi ini akan memetakan setiap karakter pada alfabet A menjadi integer dan disimpan pada suatu larik. Nilai integer tersebut merupakan nilai kemunculan karakter terakhir pada *pattern* P. Misal kita mempunyai alfabet A dan *pattern* P sebagai berikut.

A = {a, b, c, d, e}

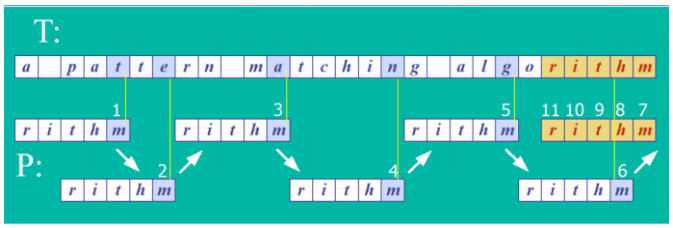
P: “aababcdae”

Maka *last occurrence function* yang dibuat adalah sebagai berikut.

x	a	b	c	d	e
L(x)	7	4	5	6	8

Tabel 2.5.1 Contoh Perhitungan *Last Occurrence Function* Pattern P dan Alfabet A
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (diakses pada 8 Mei 2021)

Setelah mengeksekusi *last occurrence function*, algoritma BM akan melakukan pencocokan string dengan menggunakan kedua teknik diatas. Berikut adalah contoh pencocokan string dengan teks "a pattern matching algorithm" dan *pattern* yang akan dicari "rithm".



Gambar 2.5.1 Ilustrasi Pencocokan String dengan Algoritma BM
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (diakses pada 8 Mei 2021)

Berdasarkan ilustrasi di atas, pergeseran *pattern* dilakukan dengan cara mengeksekusi teknik *looking glass* dan *character jump* berdarkan *least occurrence function*.

Kompleksitas waktu rata-rata yang digunakan untuk mengeksekusi algoritma Boyer-Moore adalah O(n) sementara kompleksitas waktu terburuk sebesar O(nm+A). Algoritma ini lebih cepat apabila dibandingkan dengan *brute force* dan cocok digunakan untuk mencari teks dengan alfabet yang beragam.

F. *Jaro Similarity*

Selain algoritma untuk melakukan pencarian pada string, terdapat juga algoritma yang digunakan untuk mengecek kesamaan/*similarity* antara dua buah string. Salah satu algoritma ini adalah algoritma Jaro Similarity. Algoritma ini menghitung kesamaan antara dua buah string dan menghasilkan nilai antara 0 dan 1. Nilai 1 berarti kedua buah string tersebut sama persis dan nilai 0 berarti kedua buah string tersebut tidak mirip sama sekali.

Jaro Similarity dihitung menggunakan formula sebagai berikut:

$$Jaro\ similarity = \begin{cases} 0, \text{ if m=0} \\ \frac{1}{3}\left(\frac{m}{|s1|} + \frac{m}{|s2|} + \frac{m-t}{m}\right), \text{ for m!=0} \end{cases}$$

Gambar 2.6.1 Perhitungan Jaro Similarity
Sumber:
https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/ (diakses pada 9 Mei 2021)

Dengan:
m = banyaknya karakter yang sama
t = setengah dari banyaknya *transposition* (setengah dari banyaknya karakter yang cocok pada kedua buah string namun pada urutan yang berbeda)
|s1| dan |s2| = panjang dari string s1 dan s2

Dengan menggunakan jaro similarity, kita dapat menentukan similaritas antara dua buah string. Contohnya s1 = "arnab" dan s2 = "raanb", maka perhitungan jaro similarity adalah sebagai berikut [2]:
m = 5
t = ½ * 4 = 2
|s1| dan |s2| = 5
Jaro Similarity = (1/3) * {(5/5) + (5/5) + (5-2)/5 } = 0.86667

G. *Gartic.io*

Gartic.io merupakan salah satu permainan *online multiplayer* yang dibuat oleh Gartic. Permainan ini merupakan permainan tebak gambar dimana beberapa orang berusaha menebak gambar yang telah digambar oleh satu orang penggambar. Permainan ini dapat terdiri dari 5 hingga 50 pemain yang berada pada ruangan yang sama. Pemain ini secara bergantian akan menjadi penggambar pada setiap ronde dan pemain sisanya bertugas untuk menebak jawaban atas apa yang telah digambar oleh penggambar. Apabila pemain berhasil menebak kata yang digambar secara tepat, maka pemain akan mendapat poin. Apabila salah, mereka harus menebak sampai jawaban mereka benar atau waktu ronde tersebut sudah berakhir. Apabila jawaban yang dimasukkan oleh penebak hampir mendekati benar, maka penebak akan diberikan pesan bahwa jawaban yang barusan mereka masukkan mendekati jawaban yang benar.

Gambar 2.7.1 Tampilan Permainan Gartic.io
Sumber:
https://gartic.io/ (diakses pada 9 Mei 2021)

III. Analisis dan Pembahasan

Pada bab ini, penulis akan mengimplementasikan algoritma *brute force*, Knuth-Morris-Pratt, dan Boyer-Moore pada bahasa pemrograman python. Penulis juga akan membuat beberapa kasus uji yang dapat merepresentasikan permainan Gartic.io pada *Command Line Interface* (CLI) dengan menggunakan bahasa pemrograman python. Untuk melakukan perhitungan jaro similarity antara input pengguna dan jawaban sebenarnya pada program, penulis menggunakan salah satu *library* pada python yang bernama python-Levenshtein. Berikut beberapa implementasi dan kasus uji yang diterapkan pada bahasa pemrograman python.

A. Penerapan Algoritma Brute Force

```
def bfMatch(text : str, pattern : str):
    n = len(text)
    m = len(pattern)
    for i in range ((n-m)+1):
        j = 0
        while((j<m) and (text[i+j] ==
pattern[j])):
            j+=1
        if (j==m):
            return True
    return False
```

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

```
def computeFail(pattern : str):
    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

def kmpMatch(text : str, pattern : str):
    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 True # Found
            i+=1
            j+=1
        elif(j>0):
            j = fail[j-1]
        else:
            i+=1
    return False # Not Found
```

C. Penerapan Algoritma Boyer-Moore (BM)

```
def lastOccurence(pattern):
    chars = [-1]*128 #ASCII is 128

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

    return chars

def bmMatch(text : str, pattern : str):
    lastOccurences = lastOccurence(pattern)
    n = len(text)
    m = len(pattern)
    i = m-1
    if(i > n-1):
        return False
    j = m-1
    while(i <= (n-1)):
        if(pattern[j] == text[i]):
            if(j == 0):
                return True
            else:
                i-=1
                j-=1
        else:
            lo = lastOccurences[ord(text[i])]
            i = i+m-min(j, 1+lo)
            j = m-1
    return False
```

D. Penerapan Algoritma Jaro Similarity

```
from Levenshtein import jaro

def checkSimilarity(text, match):
    return jaro(text, match)
```

E. Pengujian Waktu Eksekusi yang Diperlukan Tiap Algoritma

Penulis melakukan pengujian terhadap waktu eksekusi untuk masing-masing algoritma dengan menggunakan bahasa pemrograman python dan *library* time yang disediakan. Penulis membuat *mockup* permainan Gartic.io pada *Command Line Interface* (CLI) dengan menyediakan gambar pensil dan kolom jawaban yang akan diisi oleh pengguna program. Jawaban yang diberikan merupakan jawaban yang benar (kata "pensil") dan kemudian untuk setiap algoritma pencocokan string (*brute force*, KMP, dan BM), penulis membuat main program yang berbeda untuk ketiga algoritma tersebut dan tiap main program menghitung waktu yang diperlukan untuk mengeksekusi fungsi dari tiap-tiap algoritma yang sudah didefinisikan di atas.

```
Jawaban: pensil
Brute force : 12900 ns
```

Gambar 3.5.1 Contoh Hasil Eksekusi Progam Pengujian Algoritma *Brute Force*

Percobaan	Waktu Algoritma *Brute Force* (ns)	Waktu Algoritma KMP (ns)	Waktu Algoritma BM (ns)
1	13000	20900	18000
2	12300	20300	18200
3	8900	21600	21000
4	12900	16500	18900

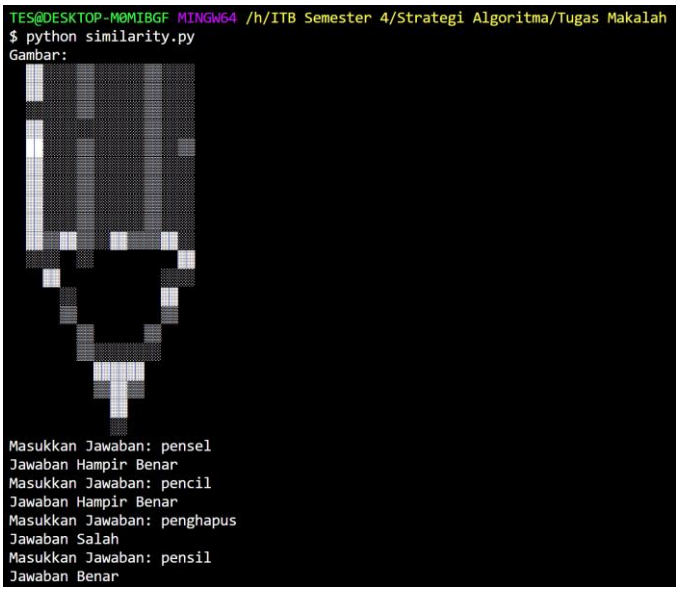
Tabel 3.5.1 Perbandingan Waktu Algoritma *Brute Force*, KMP, dan BM

Tabel di atas merupakan tabel yang menjelaskan tentang waktu eksekusi masing-masing algoritma pada tiap percobaan. Pada percobaan pertama hingga keempat, algoritma *brute force* membutuhkan waktu eksekusi sebesar 13000, 12300, 8900, dan 12900 ns. Pada percobaan pertama hingga keempat, algoritma KMP membutuhkan waktu eksekusi sebesar 20900, 20300, 21600, dan 16500. Pada percobaan pertama hingga keempat, algoritma BM membutuhkan waktu eksekusi sebesar 18000, 18200, 21000, dan 18900. Ketika dihitung rata-rata dari setiap algoritma, terhitung bahwa rata-rata waktu eksekusi algoritma *brute force* sebesar 11775 ns, algoritma KMP sebesar 19825 ns, dan algoritma BM sebesar 19025 ns.

F. Pengujian Algoritma Jaro Similarity

Penulis melakukan pengujian terhadap algoritma jaro similarity dengan menggunakan bahasa pemrograman python yang digabungkan dengan *library* python-Levensthein. Penulis membuat *mockup* permainan Gartic.io dengan menyediakan gambar pensil. Kemudian program akan meminta masukan jawaban pengguna. Apabila jawaban benar yang diindikasikan dengan pencocokan dengan algoritma *brute force*, KMP, dan BM ketiganya benar, maka program akan berhenti dan mengeluarkan *output* "Jawaban Benar". Namun, apabila masukan masih salah, masukan pengguna tersebut awalnya dihitung dengan menggunakan algoritma jaro similarity. Apabila hasil perhitungan dengan menggunakan jaro similarity lebih dari sama dengan 0.8, maka program akan memberikan *output* "Jawaban Hampir Benar" lalu meminta masukan pengguna lagi. Apabila hasil perhitungan kurang dari 0.8, maka program akan mengeluarkan *output* "Jawaban Salah" lalu meminta masukan pengguna lagi.

Penulis memberikan empat masukan ketika melakukan eksekusi program. Masukan pertama berupa kata "pensel" (jawaban salah namun mendekati), masukan kedua berupa kata "pencil" (jawaban salah namun mendekati), masukan ketiga berupa kata "penghapus" (jawaban salah dan tidak mendekati) dan masukan keempat berupa kata "pensil" (jawaban benar).



Gambar 2.6.1 Eksekusi Program yang Memanfaatkan Jaro Similarity

Dari hasil eksekusi program di atas, terlihat bahwa keluaran program ketika pengguna memasukkan kata "pensel" dan "pencil" adalah "Jawaban Hampir Benar", keluaran ketika pengguna memasukkan kata "penghapus" adalah "Jawaban Salah", dan keluaran ketika pengguna memasukkan kata "pensil" adalah "Jawaban Benar".

G. Pembahasan Pengujian

Berdasarkan hasil uji coba terhadap waktu eksekusi yang diperlukan untuk menjalankan algoritma *brute force*, KMP, dan BM, terlihat bahwa waktu rata-rata yang diperlukan oleh algoritma *brute force* sebesar 11775 ns, algoritma KMP sebesar 19825 ns, dan algoritma BM sebesar 19025 ns. Berdasarkan waktu rata-rata di atas, terlihat bahwa penggunaan algoritma *brute force* memberikan rata-rata waktu paling sedikit. Hal ini disebabkan karena algoritma *brute force* melakukan pencocokan string secara langsung tidak seperti KMP dan BM yang melakukan *preprocessing* terhadap *pattern* yang diberikan. Karena pencocokan string ini dilakukan pada teks dan *pattern* yang relatif kecil ("pensil" dengan panjang teks 6), pendekatan *brute force* merupakan pendekatan yang paling efektif untuk menjalankan permainan tebak gambar seperti Gartic.io.

Berdasarkan hasil uji coba terhadap algoritma jaro similarity, terlihat bahwa *output* program yang dihasilkan terhadap berbagai masukan jawaban benar, jawaban mendekati benar, dan jawaban salah sudah sesuai. Program akan memberikan *output* berupa "Jawaban Hampir Benar" apabila masukan yang dimasukkan pengguna memiliki kesamaan terhadap kunci jawaban sebesar lebih dari sama dengan 0.8. Hal ini membuktikan bahwa algoritma jaro similarity juga dapat digunakan pada permainan Gartic.io untuk memberikan petunjuk apabila jawaban yang dimasukkan pengguna sudah mendekati jawaban sebenarnya.

IV. KESIMPULAN

Algoritma pencocokan string dapat digunakan pada permainan Gartic.io untuk mencocokkan masukan pengguna dengan jawaban sebenarnya. Berdasarkan proses pengujian, algoritma terbaik untuk melakukan pencocokan string pada permainan Gartic.io adalah algoritma *brute force* karena teks dan *pattern* yang dicocokkan panjangnya relatif sedikit sehingga algoritma *brute force* dianggap paling baik untuk diterapkan pada game karena tidak melakukan proses *preprocessing* seperti algoritma KMP dan BM. Selain algoritma pencocokan string, terdapat juga algoritma jaro similarity yang digunakan untuk menghitung kesamaan antara dua buah string yang dapat digunakan pada permainan Gartic.io untuk mengecek apakah masukan pengguna sudah mendekati jawaban yang sebenarnya.

V. SARAN

Saran yang dapat diberikan dari penulisan makalah ini adalah penulis dapat membuat program dengan tampilan yang lebih menyerupai Gartic.io agar pembaca mengetahui pemanfaatan algoritma secara lebih mendetail. Selain itu, terdapat beberapa algoritma lain selain jaro similarity untuk membandingkan kesamaan antara dua buah string sehingga sebaiknya penulis juga membandingkan dari berbagai macam algoritma tersebut.

LINK VIDEO PADA YOUTUBE

Video youtube yang dibuat dapat dilihat pada pranala berikut:

https://youtu.be/jDStpNNNLDw

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa karena atas bantuan-Nya, penulis dapat menyelesaikan makalah ini dengan baik. Penulis juga mengucapkan terima kasih kepada kedua orang tua yang telah mendukung dalam penulisan makalah ini. Penulis juga mengucapkan terima kasih kepada dosen pengajar mata kuliah IF2211 Strategi Algoritma kelas K03 Bapak Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc.,Ph.D. karena atas ilmu yang telah diajarkan dan bimbingan beliau, penulis dapat menyelesaikan makalah ini dengan baik. Penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. karena telah menyediakan materi-materi terkait pada pranala pembelajaran Strategi Algoritma yang dapat digunakan sebagai sumber belajar dalam pembuatan makalah ini. Terakhir, penulis mengucapkan terima kasih kepada teman-teman jurusan Teknik Informatika 2019 karena telah memberikan dukungan dan bantuan pada pengerjaan makalah ini.

REFERENCES

[1] Levitin, A., 2012. *Introduction to the design & analysis of algorithms*. 3rd ed. Boston: Pearson.

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
Diakses pada tanggal 7 Mei 2021

[3] https://betterprogramming.pub/what-every-programmer-should-know-about-string-a6611537f84e
Diakses pada tanggal 8 Mei 2021

[4] http://cs.indstate.edu/~kmandumula/presentation.pdf
Diakses pada tanggal 8 Mei 2021

[5] http://cs.indstate.edu/~soddiraju/abstract.pdf
Diakses pada tanggal 8 Mei 2021

[6] https://www.geeksforgeeks.org/jaro-and-jaro-winkler-similarity/
Diakses pada tanggal 9 Mei 2021

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.

Surabaya, 11 Mei 2021

Fadel Ananda Dotty
13519146