Penerapan Algoritma Pencocokan String pada Penilaian Jawaban Pendek

Muhammad Dzaki Razaan Faza 13519033
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519033@std.stei.itb.ac.id

***Abstrak*—Pandemi COVID-19 memaksa seluruh elemen masyarakat untuk melakukan aktivitas di dalam rumah tidak terkecuali institusi pendidikan. Seluruh aktivitas belajar dan mengajar harus dilakukan secara daring termasuk juga ujian. Dengan diterapkannya metode ujian *online* tentu membawa banyak pengalaman baru dalam dunia pendidikan. Dengan digunakannya *file* teks untuk menyimpan jawaban dibandingkan kertas membuka banyak kemungkinan baru. Salah satunya adalah penerapan algoritma pencocokan *string* untuk menilai jawaban pendek siswa dalam ujian. Dengan menggunakan algoritma Levenshtein Distance untuk mengukur kemiripan jawaban siswa dan kunci jawaban sebuah program dapat menilai apakah jawaban siswa benar atau tidak. Setelah dilakukan analisis ditemukan bahwa metode ini menghasilkan keputusan yang cukup akurat. Walaupun begitu, masih perlu dilakukan penelitian dan pengujian lebih lanjut sebelum benar-benar diterapkan dalam sistem penilaian ujian siswa yang sesungguhnya.**

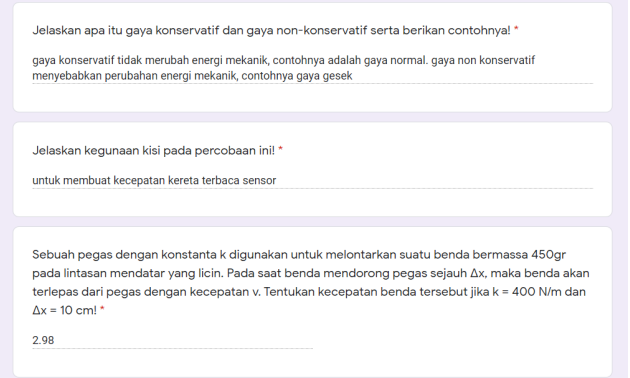
Keywords—jawaban pendek; esai; essay; grading; penilaian; string matching; levenshtein distance; koreksi

I. PENDAHULUAN

Pandemi COVID-19 memaksa keseluruhan aktivitas untuk dilaksanakan secara daring. Tidak terkecuali institusi pendidikan juga dipaksa untuk melakukan pembelajaran dari rumah demi menjaga kesehatan dan keamanan para siswa. Namun, pandemi ini tidak boleh membatasi siswa dalam memperoleh kualitas edukasi yang baik. Salah satu potensi yang dapat dikembangkan dengan adanya sistem daring ini adalah efektivitas dan efisiensi dari pengoreksian jawaban soal ujian atau ulangan harian.

Sebelum adanya pandemi, ujian selalu diadakan dengan menggunakan kertas dan pensil atau pena. Metode tertulis ini memiliki beberapa kekurangan salah satunya adalah terdapat kesulitan dalam melakukan koreksi atau penilaian pada jawaban pendek atau esai karena tulisan yang tidak jelas dan perlu adanya pengkoreksian secara manual dari guru atau dosen sehingga memperlama proses penilaian.

Sedangkan saat ini digunakan *platform* seperti *google form* atau *website* kuliah untuk memfasilitasi hal tersebut. Pada metode daring kesulitan membaca kesulitan dapat dihilangkan karena format penulisan jawaban selalu sama. Selain itu, karena jawaban disimpan dalam bentuk *file* memungkinkan adanya otomasi pengecekan jawaban sehingga nilai bisa didapat secara lebih efektif dan efisien. Makalah ini akan membahas metode yang dapat dilakukan untuk melakukan pengecekan jawaban pendek dalam soal ujian menggunakan algoritma *string matching* menggunakan *levenshtein distance.*



Gambar 1.2 Ujian *Online*

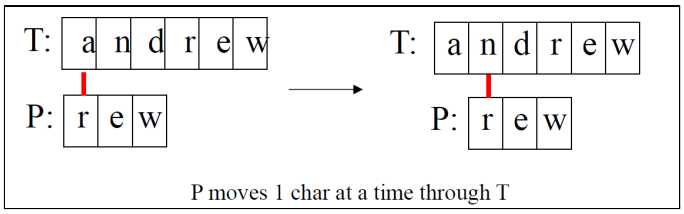
II. LANDASAN TEORI

A. *String / Pattern Matching*

String/pattern matching adalah proses mencari lokasi pertama dalam teks yang bersesuaian dengan pattern. Teks merupakan string dengan panjang n karakter. Pattern adalah string dengan panjang m (dan m sangat kecil dibandingkan n) yang akan dicari dalam teks. Penerapan string/pattern matching sering ditemukan dalam kehidupan sehari-hari, seperti pencarian dalam text editor, web search engine, analisis citra, bioinformatika, dan masih banyak lagi. Dalam melakukan string/pattern matching terdapat beberapa algoritma yang bisa digunakan. Diantaranya adalah algoritma brute force, algoritma Knuth-Morris-Pratt(KMP), dan algoritma Boyer-Moore(BM), *regular expression* dan Levenshtein Distance.

B. *Algoritma Brute-Force*

Algoritma brute force merupakan algoritma string matching yang paling sederhana dan menyelesaikan permasalahan secara gamblang. Proses pencarian pattern dalam teks dilakukan dengan cara mengecek setiap posisi dalam teks T apakah pattern P dimulai dari posisi tersebut.

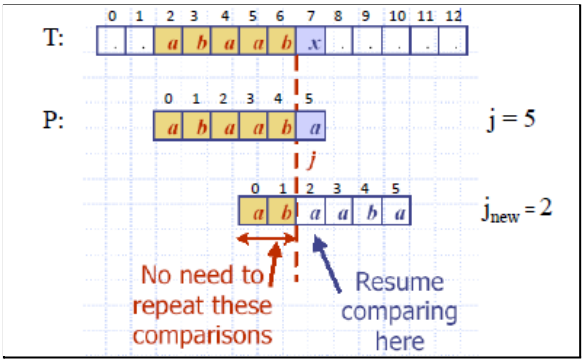


Gambar 2.1 Ilustrasi Algoritma Brute Force

Kompleksitas algoritma ini dapat dilihat dari jumlah perbandingan karakter yang dilakukan. Dalam berbagai kasus kompleksitasnya dapat mencapai O(mn), O(n), O(m+n) untuk kasus terburuk, terbaik, dan rata-rata. Algoritma brute force akan sangat cepat apabila alfabet dari teks besar (contoh: A..Z, a..z, 1..9) dan akan sangat lambat bila alfabetnya kecil (contoh: 0,1 dalam binary file).

C. Algoritma Knuth-Morris-Pratt

Algoritma KMP mencari pattern dalam teks dengan urutan kiri ke kanan seperti algoritma brute force, tetapi menggerakkan pattern dengan lebih "pintar". Jika ada ketidakcocokan antara teks T dan pattern P di P[j] (karakter P ke-j), akan dilakukan pergeseran pattern sehingga sebisa mungkin menghindar perbandingan yang tidak berguna. KMP menjawab cara ini dengan mencari prefiks terbesar dari P[0-j-1] yang merupakan sufiks dari P[1..j-1] menggunakan fungsi pinggiran.

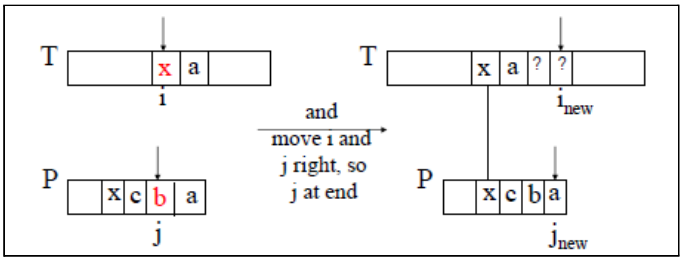


Gambar 2.2 Ilustrasi Algoritma KMP

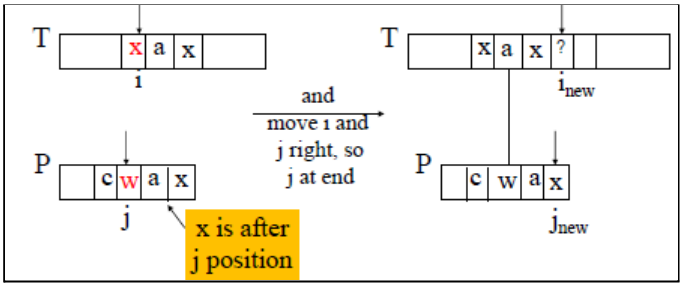
Kompleksitas waktu dari algoritma KMP ditentukan dari perhitungan fungsi pinggiran yaitu O(m) dan pencarian string yaitu O(n). Secara keseluruhan kompleksitas algoritma ini adalah O(m+n). Hasil ini membuktikan bahwa algoritma KMP lebih cepat dari brute force dalam melakukan string matching. Algoritma KMP tidak akan pernah membutuhkan pergerakan mundur di teks masukan sehingga sangat cocok untuk memproses file sangat besar yang dibaca dari perangkat eksternal atau melalui arus jaringan. Namun, algoritma KMP semakin tidak efektif dengan bertambahnya ukuran dari alfabet. Kemungkinan ketidakcocokan menjadi bertambah dan ketidakcocokan biasanya terjadi di awal pattern padahal algoritma KMP lebih cepat jika ketidakcocokan terjadi di belakang pattern.

D. Algoritma Boyer-Moore

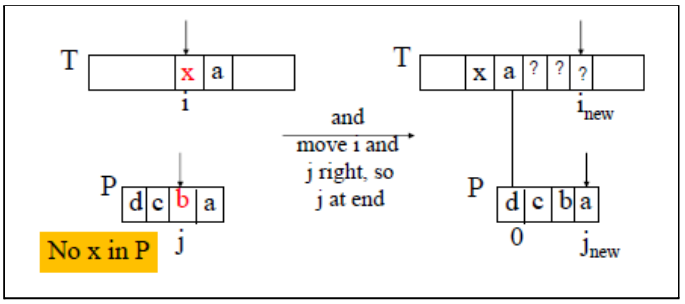
Algoritma BM menerapkan string matching dengan dua teknik dasar. Teknik pertama adalah dengan mencari pattern P dalam teks T dengan bergerak secara mundur dalam P, memulai perbandingan karakter dari akhir ke awal pattern. Teknik kedua adalah character-jump yaitu ketika terjadi ketidakcocokan antara P[j] dan T[i] pada T[i]=x maka akan terjadi tiga kasus yang mungkin. Kasus pertama adalah jika P mengandung x maka geser P ke kanan hingga kemunculan x terakhir di P sejajar dengan T[i] lalu mulai perbandingan kembali. Kasus kedua adalah jika dengan menggeser tidak bisa mensejajarkan x terakhir di P dengan T[i] maka geser P ke kanan sebanyak 1 karakter ke T[i+1]. Kasus ketiga adalah kemungkinan kejadian yang tidak ditangani kasus pertama dan kedua maka geser P sehingga P[0] sejajar dengan T[i+1].



Gambar 2.3 Ilustrasi Algoritma MP Kasus 1



Gambar 2.4 Ilustrasi Algoritma MP Kasus 2

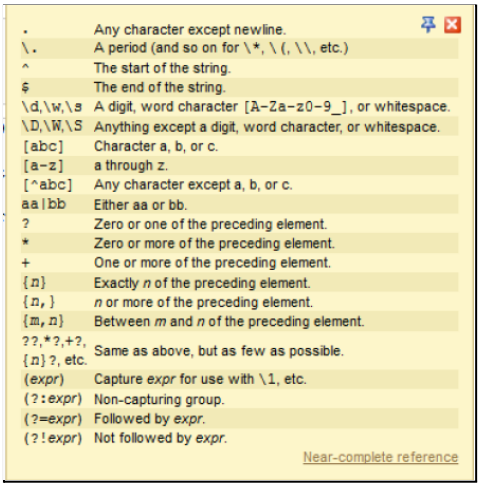


Gambar 2.5 Ilustrasi Algoritma MP Kasus 3

Algoritma BM mempunyai kompleksitas algoritma dalam kasus terburuk yaitu O(nm+A). Namun, algoritma ini lebih cepat ketika ukuran alfabet(A) besar dan lambat jika ukurannya kecil. Algoritma BM akan sangat cepat dibandingkan brute force untuk pencarian pattern dalam teks bahasa inggris.

E. *Regular Expression (Regex)*

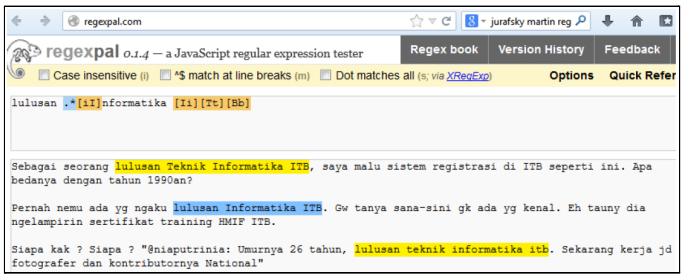
Regex adalah serangkaian karakter yang mendefinisikan sebuah pola pencarian. Pola tersebut digunakan dalam string matching untuk melakukan operasi "cari" atau "cari dan ganti" pada string, atau untuk memeriksa string masukan. Ekspresi reguler merupakan teknik yang dikembangakan dalam bidang ilmu komputer teori dan teori bahasa formal.



.	Any character except newline.
\.	A period (and so on for *, \(, \\, etc.)
^	The start of the string.
$	The end of the string.
\d,\w,\s	A digit, word character [A-Za-z0-9_], or whitespace.
\D,\W,\S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	a through z.
[^abc]	Any character except a, b, or c.
aa|bb	Either aa or bb.
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{*n*}	Exactly *n* of the preceding element.
{*n*,}	*n* or more of the preceding element.
{*m*,*n*}	Between *m* and *n* of the preceding element.
??,*?,+?, {*n*}?, etc.	Same as above, but as few as possible.
(*expr*)	Capture *expr* for use with \1, etc.
(?:*expr*)	Non-capturing group.
(?=*expr*)	Followed by *expr*.
(?!*expr*)	Not followed by *expr*.

Near-complete reference

Gambar 2.6 Notasi Regex



Gambar 2.7 Ilustrasi String Matching menggunakan Regex

F. *Levenshtein Distance*

Levenshtein distance adalah “jarak ” di antara dua *string*. “Jarak” yang dimaksud adalah jumlah minimum perubahan karakter yang harus dilakukan agar salah satu *string* dapat diubah menjadi *string* yang lain. Perubahan karakter ini didefinisikan sebagai penambahan, penghapusan, atau substitusi dari satu karakter atau pertukaran posisi dua karakter dalam *string*.

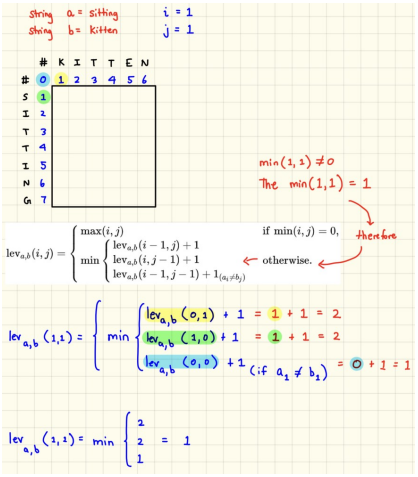
Sebagai contoh levenshtein distance dari kata “kitten” dan “sitting” adalah tiga, artinya dibutuhkan minimum tiga perubahan untuk melakukan transformasi dari kitten ke sitting atau sebaliknya. Perubahan tersebut adalah:

1. **k**itten → **s**itten (substitsui k dengan s)
2. sitten → sitt**i**n (substitusi e dengan i)
3. sittin → sitting (penambahan g di akhir kata)

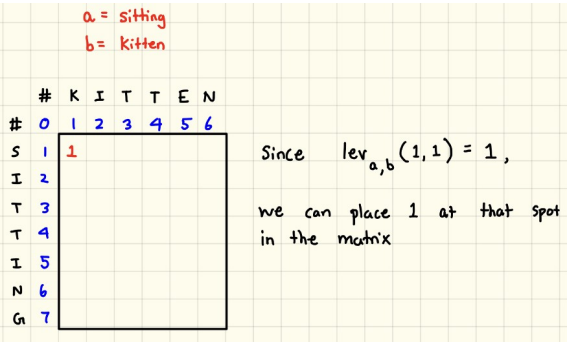
Formula dari levenshtein distance ini adalah:

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

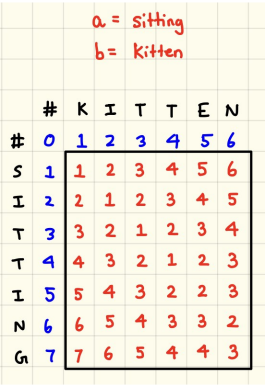
Formula ini diterapkan pada sebuah matrik dengan baris sepanjang *string 1* dan kolom sepanjang *string* 2. Berikut adalah contoh langkah penyelesaian levenshtein distance dari kitten dan sitting.



Gambar 2.8 Langkah Penyelesaian Kemiripan *String* Menggunakan Levenshtein Distance I



Gambar 2.9 Langkah Penyelesaian Kemiripan *String* Menggunakan Levenshtein Distance II



	#	K	I	T	T	E	N
#	0	1	2	3	4	5	6
S	1	1	2	3	4	5	6
I	2	2	1	2	3	4	5
T	3	3	2	1	2	3	4
T	4	4	3	2	1	2	3
I	5	5	4	3	2	2	3
N	6	6	5	4	3	3	2
G	7	7	6	5	4	4	3

Gambar 2.10 Hasil Matriks Levenshtein Distance

Dari matriks tersebut didapat bahwa levenshtein distance dari string kitten dan sitting adalah tiga ditunjukkan dari elemen terakhir matriks yaitu elemen paling kanan bawah. Untuk menghitung kemiripan dari kedua *string* maka akan dilakukan perhitungan dengan rumus:

similarity = levenshtein distance / max(panjang string 1, panjang string 2)

Pada kasus ini maka tingkat kemiripan dari *string* kitten dan sitting adalah 3 / max(6, 7) = 3 / 7 = 0.429 atau 42,9%. Jika diasumsikan kedua *string* dikatakan mirip untuk tingkat kemiripan di atas 70% maka *string* kitten dan sitting tidak sama.

G. *Stopword Removal and Stemming*

Dalam NLP (*Natural Language Processing*) stop word merupakan kata yang diabaikan dalam pemrosesan, kata-kata ini biasanya disimpan ke dalam *stop lists*. Karakteristik utama dalam pemilihan *stop word* biasanya adalah kata yang mempunyai frekuensi kemunculan yang tinggi misalnya kata penghubung seperti “dan”, “atau”, “tapi”, “akan” dan lainnya. Tidak ada aturan pasti dalam menentukan *stop word* yang akan digunakan, penentuan *stop word* bisa disesuaikan dengan kasus yang sedang diselesaikan. Tujuan utama dalam penerapan proses *Stopword Removal* adalah mengurangi jumlah kata dalam sebuah dokumen yang nantinya akan berpengaruh dalam kecepatan dan performa dalam kegiatan NLP.

Sedangkan *stemming* adalah pemotongan imbuhan pada kata berimbuhan yang dijalankan dengan algoritma tertentu. Dari kata menghasilkan menjadi hasil, kata membeli jadi beli, dan masih banyak contoh lain.

Tujuan utama dari dilakukannya *stopword removal dan stemming* pada kasus pengecekan jawaban pendek dalam ujian adalah untuk menyaring kalimat menjadi inti sari katanya sehingga tidak ada jawaban yang bertele-tele dan proses pencocokan dapat dilakukan dengan lebih mudah dan cepat karena terjadi penyaringan kata.

III. IMPLEMENTASI DAN PENGUJIAN

A. *Implementasi*

Implementasi dilakukan dengan menggunakan bahasa pemrograman Python dan memanfaatkan *library* Sastrawi untuk *stopword removal* dan *stemming*. Metode pencocokan *string* yang digunakan adalah Levenshtein Distance sehingga pencocokan tidak hanya sekedar sama atau tidak melainkan juga tingkat kemiripannya. Proses penilaian / *grading* diterapkan dengan mempertimbangkan tingkat kemiripan kunci jawaban dibandingkan dengan jawaban siswa.

Berikut adalah implementasi secara detail algoritma Levenshtein Distance pada proses penilaian jawaban pendek / esai siswa dalam ujian:

1. Membersihkan *query* kunci jawaban dan jawaban siswa dari kata sambung (*stopword*) dan menyaring *query* menjadi kata dasar

```
def stemming(query):
    #Proses Stemming dokumen

    #Membuat Stemmer
    factory = StemmerFactory()
    stemmer = factory.create_stemmer()

    #Stemming
    output = stemmer.stem(query)
    return output

def filtering_stopword(query):
    #Membersihkan stopword
    factory = StopWordRemoverFactory()
    stopword = factory.create_stop_word_remover()
    output = stopword.remove(query)
    return output

def stemming_filtering_query(query):
    #Stemming query dan filtering stopword
    query_clean = filtering_stopword(query)
    query_stemmed = stemming(query_clean)
    return query_stemmed.split()
```

2. Menghitung kemiripan masing-masing kata dalam jawaban siswa dengan kunci jawaban menggunakan Levenshtein Distance dan hasilnya disimpan dalam sebuah matriks kemiripan

```
def lev(typo, bener):
    #Menghitung Levenshtein Distance dan tingkat kemiripan kedua string
    typo1 = "#" + typo
    bener1 = "#" + bener
    matriks = [[0 for i in range(len(bener1))]for j in range(len(typo1))]
    for i in range(len(typo1)):
        for j in range(len(bener1)):
            if(min(i,j) == 0):
                matriks[i][j] = max(i,j)
            else:
                a = matriks[i-1][j] + 1
                b = matriks[i][j-1] + 1
                c = matriks[i-1][j-1]
                if(typo1[i] != bener1[j]):
                    c+=1
                matriks[i][j] = min(a,b,c)
    distance = matriks[len(typo1)-1][len(bener1)-1]
    return distance / max(len(typo1)-1, len(bener1)-1)

def simMatrix(arr1, arr2):
    #Mengembalikan array matriks kemiripan dari kedua input string
    similar = [[0 for i in range(len(arr2))] for j in range(len(arr1))]
    for i in range(len(arr1)):
```

```
    for j in range(len(arr2)):
        similar[i][j] = lv.lev(arr1[i],arr2[j])
return similar
```

3. Mencari tingkat kemiripan maksimum antara satu kata dalam jawaban siswa dengan seluruh kata dalam kunci jawaban. Skala kemiripan yang dibuat di program ini adalah 0 artinya mirip dan 1 artinya tidak mirip sama sekali

```
def maxSim(Matrix):
    sim = []
    for i in Matrix:
        sim.append(min(i))
    return sim
```

4. Menghitung rata-rata kemiripan jawaban siswa dengan kunci jawaban lalu memberikan penilaian berdasarkan angka tersebut. Pada implementasi ini ditentukan bahwa bila tingkat kemiripannya adalah >75% (atau angka kemiripannya <0.25) maka jawaban siswa dianggap benar

```
def grading(sim):
    sum = 0
    for i in sim:
        sum += i
    avg = sum / len(sim)
    if avg < 0.25:
        return "benar"
    else:
        return "salah"
```

Contoh dari pemrosesan *query* tersebut adalah sebagai berikut. Misal :

Kunci Jawaban :

Makhluk hidup terdiri dari tiga yaitu manusia, hewan, dan tumbuhan

Jawaban Siswa :

makhluk hidup hanya ada manusia

Query bersih :

Kunci Jawaban :

['makhluk', 'hidup', 'diri', 'tiga', 'manusia', 'hewan', 'tumbuh']

Jawaban Siswa :

['makhluk', 'hidup', 'ada', 'manusia']

Matriks Kemiripan :

	makhluk	hidup	ada	manusia
makhluk	0.00	0.85	0.85	0.71
hidup	0.85	0.00	0.80	0.85
diri	1.00	0.80	1.00	0.85
tiga	1.00	0.80	0.75	0.85
manusia	0.71	0.85	0.71	0.00
hewan	1.00	0.80	0.80	1.00
tumbuh	0.85	0.83	1.0	1.0

Maksimum Kemiripan :

makhluk : 0 (mirip)
hidup : 0 (mirip)
diri : 0.8
tiga : 0.75
manusia : 0 (mirip)
hewan : 0.8
tumbuh : 0.83

Hasil Penilaian :

Jawaban dari siswa tersebut adalah salah

B. Pengujian

1. Kasus I -- Jawaban Sederhana dan Pendek

```
Masukkan kunci jawaban:
-->Makhluk hidup terdiri dari tiga yaitu manusia, hewan, dan tumbuhan

Masukkan jawaban yang ingin dikoreksi:
-->makhluk hidup hanya ada manusia

Jawaban dari siswa tersebut adalah salah
```

2. Kasus II -- Jawaban Bertele-tele

Masukkan kunci jawaban:
-->Panjang minimalnya adalah 144 cm

Masukkan jawaban yang ingin dikoreksi:

-->Diketahui: Panjang sisi belah ketupat: 36 cm
Ditanya: Berapakah panjang minimal pita yang diperlukan kakak untuk menghias kado?
Jawab:Keliling belah ketupat:4 x sisi = 4 x 36 cm = 144 cmJadi panjang minimal yang diperlukan adalah 144 cm atau 1,44 meter.

Jawaban dari siswa tersebut adalah benar

IV. ANALISIS

Berdasarkan implementasi dan hasil pengujian ditemukan bahwa metode penilaian jawaban ujian dengan menggunakan Levenshtein Distance dapat menghasilkan keputusan yang cukup akurat baik untuk jawaban singkat ataupun jawaban panjang bertele-tele dengan inti yang sama. Namun, tentunya masih ada keterbatasan dalam metode pengujian sehingga mungkin masih banyak kasus yang belum dicakup. Selain itu, metode ini masih bisa dikembangkan dengan memberikan penilaian berdasarkan tingkat akurasinya misal rentang 75-100% bernilai 5, 65-75% bernilai 4, dan seterusnya. Oleh karena itu, metode ini masih perlu diteliti dan dikembangkan lagi sebelum bisa benar-benar diimplementasikan pada sistem sesungguhnya di institusi-institusi pendidikan yang ada.

V. SIMPULAN

A. *Simpulan*

Penerapan algoritma *string matching* dan kemiripan *string* pada penilaian jawaban pendek dalam soal ujian adalah dengan cara menyaring kalimat jawaban dan kunci jawaban menggunakan *stopword removal* dan *stemming* lalu diterapkan algoritma *string matching* untuk menghitung tingkat kemiripan jawaban dengan kunci jawaban menggunakan levenshtein distance.

Penerapan algoritma Levenshtein Distance menghasilkan penilaian yang cukup akurat. Namun, seperti yang sudah dijelaskan dalam sub bab analisis masih dibutuhkan pematangan dan perbaikan metode atau dengan menggunakan pendekatan algoritma baru agar bisa diimplementasi secara nyata.

B. *Saran*

Penulis menyadari bahwa dalam pengerjaan makalah masih banyak yang dapat dikembangkan, sebagai berikut:

1. Memperdalam riset dan studi kasus dari lebih banyak jurnal dan sumber referensi lainnya.
2. Melakukan metode perancangan implementasi dan pengujian yang lebih matang.
3. Membandingkan solusi yang ditemukan saat ini dengan alternatif solusi lain

C. *Refleksi*

Pada tugas makalah strategi algoritma ini penulis belajar untuk membuat makalah yang komprehensif dan analitik. Komentar penulis terhadap tugas makalah ini adalah walaupun cukup menantang, penulis dipaksa untuk mempelajari hal baru. Penulis percaya ilmu yang didapat saat ini sangat berguna untuk kedepannya baik untuk karir atau menyelesaikan masalah di kehidupan nyata.

PRANALA VIDEO YOUTUBE

https://youtu.be/a9vjKnBXICE

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih pada Tuhan Yang Maha Esa (YME) untuk segala rahmat yang diberikan-Nya dalam pengerjaan makalah ini sehingga dapat terselesaikan tepat waktu. Penulis juga ingin berterima kasih pada orang tua yang selalu memberikan dukungan dan teman-teman yang membantu selama proses pengerjaan makalah ini. Terakhir, penulis berterima kasih pada Dr.Ir. Rila Mandala, M.Eng., MT. yang telah membimbing penulis selama satu semester sebagai dosen pengampu IF2211 Strategi Algoritma kelas 1.

DAFTAR PUSTAKA

[1] IF2211 Strategi Algoritma. (2021). Pencocokan String (String/Pattern Matching). Program Studi Teknik Informatika STEI ITB. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

[2] Khodra, M. L. (n.d.). String Matching dengan Regular Expression. Program Studi Teknik Informatika STEI ITB. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf

[3] Gomaa, W. H., & Fahmy, A. A. (2012). Short Answer Grading Using String Similarity And Corpus-Based Similarity, 3(11). https://www.researchgate.net/publication/259181691_Short_Answer_Grading_Using_String_Similarity_And_Corpus-Based_Similarity

[4] Nam, E. (2019, February 27). Understanding the Levenshtein Distance Equation for Beginners. Retrieved May 10, 2021, from https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0

[5] P, A. Y. (2017, June 3). Stopword Removal Bahasa Indonesia dengan Python Sastrawi. Retrieved May 10, 2021, from https://devtrik.com/python/stopword-removal-bahasa-indonesia-python-sastrawi/

[6] Selvi, P., & Bnerjee, A. K. (2010, August). Automatic Short –Answer Grading System (ASAGS), 2(1). https://arxiv.org/ftp/arxiv/papers/1011/1011.1742.pdf

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.

Magelang, 10 Mei 2021

Muhammad Dzaki Razaan Faza 13519033