Kombinasi Damerau Levenshtein dan Jaro-Winkler *Distance* Untuk Koreksi Kata Bahasa Inggris

http://dx.doi.org/10.28932/jutisi.v6i2.2493

Bonifacius Vicky Indriyono ✉ #1

Sistem Informasi,STMIK Kadiri
Jl. Balowerti II/26-30 Kediri
1 bonifaciusvicky@gmail.com

Abstract — **Writing is one of the efforts made by the writer to express ideas and ideas to others. But sometimes when writing, there are many errors in typing spelling, especially English spelling, resulting in errors in capturing the meaning and meaning of the writing. To overcome this problem, we need a system that can detect word spelling errors. Damerau Levenshtein and Jaro Winkler Distance Algorithms are algorithms that can be used as a solution to detect English typing errors. From the test results, it can be concluded that the Damerau Levenshtein and Jaro-Winkler Distance are able to optimally detect word mismatches and look for similarities of words compared. The Damerau Levenshtein Distance works by finding the smallest distance value, while the Jaro-Winkler Distance works by finding the greatest proximity value of the string being compared. Using this algorithm, errors in writing the spelling of words can be minimized.**

Keywords— **Algorithm; Damerau Levenshtein; Jaro Winkler; Spelling Cheker; String Matching.**

I. PENDAHULUAN

Karya tulis menjadi salah satu sarana bagi seorang penulis dalam menyampaikan ide, gagasan dan maksud kepada orang lain sebagai pembaca. Bagaimana seorang penulis menggunakan ejaan dan tata bahasa yang baik akan sangat berpengaruh pada keindahan isi serta makna yang tersirat dalam karya tulis tersebut. Namun tidak dapat dipungkiri bahwa terkadang dalam proses pengetikan naskah banyak terdapat kesalahan ejaan, kurangnya huruf dan kesalahan lainnya sehingga menimbulkan penangkapan arti serta makna yang tidak sesuai. Untuk itulah sangat diperlukan adanya keterampilan dalam menulis yang baik. Mengoreksi kesalahan pengetikan ejaan dapat dlakukan dengan membaca kembali naskah dari awal sampai akhir dan memperbaiki kesalahan satu per satu. Tentunya langkah ini sangat tidak efisien dan membutuhkan waktu yang tidak sedikit karena harus dilakukan berkali-kali agar naskah benar-benar terbebas dari kesalahan pengetikan. Kesalahan pengetikan dikelompokkan kedalam empat jenis: kesalahan penyisipan, penghapusan, penambahan dan perubahan letak dari satu atau beberapa huruf.

Untuk menanggulangi proses koreksi yang tidak efisien seperti yang dipaparkan diatas, akan jauh lebih efektif apabila terdapat sebuah aplikasi untuk mengoreksi kesalahan pengetikan secara akurat dan mampu memberikan solusi berupa kandidat kata yang benar bagi penulis. Aplikasi tersebut dinamakan aplikasi pendeteksi ejaan. Aplikasi ini melakukan pendeteksian terhadap pengetikan ejaan kata yang salah dan mampu menampilkan saran berupa ejaan kata yang sesuai [1]. Konsep utama dari aplikasi pendeteksi ejaan adalah melakukan proses pencocokan antara dua buah string dengan beberapa teknik yang ada.

Secara garis besar, teknik pencocokan *string* dapat diklasifikasikan menjadi dua bagian yakni *exact matching* dan *inexact matching*. Menurut Sagita [2], *Exact matching* diartikan sebagai suatu teknik dalam mencocokkan kata secara benar dan sesuai berdasarkan ketepatan susunan karakter dalam sebuah *string*, sedangkan *inexact matching* merupakan teknik pencocokan *string* dengan prinsip kira-kira dimana *string* yang akan dicocokkan mempunyai kemiripan namun diantara kedua *string* tersebut terdapat rangkaian karakter yang tersusun secara berbeda. Pada kelompok *inexact matching* sendiri masih terbagi lagi menjadi 2 kelompok yakni *approximate matching* dimana kelompok ini merupakan teknik dalam mencocokan string menurut kemiripan penulisannya baik dari sisi banyaknya karakter serta urutan huruf dan *phonetic matching* yang berupa teknik dalam mencocokan *string* sesuai kemiripan ucapan. Dari dua kelompok *inexact matching* tersebut, teknik yang paling sering digunakan adalah *Approximate matching*. Teknik pencocokan ini paling banyak digunakan karena mampu mencari kata-kata yang tidak sesuai dengan kaidah bahasa yang digunakan. Selain itu teknik pencocokan ini dapat dimafaatkan untuk mendeteksi *string* yang memiliki kemiripan dari sisi penulisannya [2].

Berdasarkan masalah yang telah dipaparkan sebelumnya dan analisis yang telah dilakukan terhadap penelitian sebelumnya, maka dalam penelitian ini diajukan penelitian untuk melakukan deteksi dan koreksi kesalahan pengetikan kata dalam bahasa Inggris serta pemberian solusi untuk kata yang sesuai dengan kosakata dalam kamus bahasa Inggris. Agar proses pencocokan ini lebih akurat, maka penulis menggunakan daftar kosakata bahasa Inggris dari kamus **en_US-large.dic**. Hasil dari penelitian berupa aplikasi pendeteksi ejaan yang dapat mengakomodasi semua analisis dan tujuan akhir seperti yang ditetapkan. Sebenarnya fitur pendeteksi ejaan ini sudah tersedia di beberapa aplikasi, sebagai misal *Microsoft Word*. Namun, selain karena pengguna harus memilih kata yang salah kemudian melakukan pengecekan kata menggunakan fitur *spelling & grammar* agar didapatkan kata yang benar sedangkan pengguna kebanyakan tidak menyadari apabila melakukan kesalahan pengetikan meskipun kata yang salah sudah ditandai dengan garis merah sehingga pengguna harus mengecek kembali semua teks secara manual juga karena untuk menggunakan fitur tersebut harus dilakukan beberapa langkah seting awal penggunaan fitur *spelling & Grammar*. Maka dari hasil penelitian yang berupa aplikasi pendeteksi ejaan kata diharapkan mampu lebih memudahkan pengguna untuk melakukan deteksi kesalahan ejaan tanpa harus melalui proses seting yang panjang. dan meminimalisir adanya kesalahan pengetikan khususnya pengetikan ejaan dalam bahasa Inggris sehingga arti, makna serta informasi yang terkandung dalam naskah dapat tersampaikan dan mampu ditangkap dengan baik oleh para pembaca.

II. DASAR TEORI

Pada bagian ini akan diuraikan landasan teori yang dijadikan sebagai panduan penulisan laporan penelitian yang dilakukan.

A. Algoritma Damerau Levenshtein Distance

Menurut Puji Santoso, dkk. [3], algoritma *Damerau Levenshtein Distance* banyak digunakan untuk menentukan jarak atau jumlah terkecil dari suatu proses satu *string* menjadi *string* yang lain. proses ini digunakan sebagai sarana menentukan tingkat kemiripan antar *string*. Fungsi Dalam penerapannya algoritma *Damerau Levenshtein Distance* akan menghasilkan suatu nilai jarak edit diantara dua *string* yang dibandingkan (*string* target dan *string* sumber). Nilai jarak inilah yang akan digunakan oleh sistem untuk menentukan ada tidaknya kesalahan pengetikan pada suatu kata [4]. Menurut Yeny Rochmawati dan Retno Kusumaningrum [5], algoritma *Damerau Levenshtein* dikatakan sebagai sebuah algoritma fungsi *finite string* dari sebuah huruf/karakter ke integer, sedangkan menurut Maghfira, dkk [6] *Damerau Levenshtein Distance* adalah algoritma yang dikembangkan dari *Levenshtein Distance*. Prinsip kerja dari algoritma ini adalah dengan memastikan jumlah terkecil dari proses untuk melakukan perubahan dari satu kata menjadi kata yang lain. Menurut Sutisna dan Adisantoso [7], algoritma *Damerau Levenshtein Distance* memiliki fungsi kerja membandingkan sederet *string* dengan berpatokan pada empat jenis *typographical error* (kesalahan pengetikan). Kesalahan pengetikan tersebut meliputi: 1). Menyisipkan atau menambahkan satu huruf/karakter dari suatu *string*. 2). Menghapus atau menghilangkan satu huruf/karaker dari suatu *string*. 3). Mengganti satu huruf dengan huruf yang lain dan 4). Menukar satu huruf dengan huruf lain dimana posisi *string* yang dibandingkan memiliki posisi berurutan.

Masih menurut Sutisna dan Adisantoso [7], selain empat jenis penyebab kesalahan pengetikan yang dipaparkan diatas, kesalahan dalam ejaan juga dapat disebabkan oleh beberapa faktor berikut : a). Kekurangpahaman dalam penulisan →. Bagian ini menjelaskan bahwa kesalahan yang berupa kekurangpahaman dalam penulisan erat kaitannya dengan dengan ucapan kata dan penulisan ejaan yang sebenarnya. b). Kesalahan dalam pengetikan kata →. Kesalahan dalam kelompok ini memiliki keterkaitan dengan kesalahan jari-jari saat mengetik maupun letak dari tombol papan ketik.

Damerau Levenshtein menurut pendapat dari Sutisna dan Adisantoso (2010) juga dapat diartikan sebagai sebuah matriks jarak yang diberikan string S1,S2,S3 dimana dalam matriks tersebut memenuhi kondisi-kondisi berikut : 1). Non-negatif: d(Kata1, Kata2) $\geqslant$ 0; 2). Non-degenerasi: d(kata1, kata2) = 0 jika dan hanya jika s1 = s2; 3). Simetri: d(kata1, kata2) = d(kata2, kata1); 4). *Triangle Inequality*: d(kata1, kata2) + d(kata2, kata3) $\geqslant$ d(kata1, kata3).

Menurut Zhao [8], langkah-langkah algoritma *Damerau Levenshtein Distance* dapat dituliskan dalam bentuk *pseudocode* seperti diperlihatkan dalam Gambar 1 :

```
1:  DL(A[1 : m], B[1 : n])
2:  for j ← 0 to n do
3:      H[−1][j] ← maxVal; H[0][j] ← j
4:  end for
5:  for i ← 1 to m do
6:      H[i][−1] ← maxVal; H[i][0] ← i
7:      last_col_id ← −1
8:      for j ← 1 to n do
9:          diag ← H[i − 1][j − 1] + c(A[i], B[j])
10:         left ← H[i][j − 1] + 1
11:         up ← H[i − 1][j] + 1
12:         k = last_row_id[B[j]], l = last_col_id
13:         transpose ← H[k − 1][l − 1] + (i − k − 1) + 1 +
            (j − l − 1)
14:         H[i][j] ← min{diag, left, up, transpose}
15:         if A[i] = B[j] then
16:             last_col_id ← j
17:         end if
18:     end for
19:     last_row_id[A[i]] ← i
20: end for
21: return H[m][n]
```

Gambar 1. *Pseudocode* algoritma *damerau levenshtein distance*

Berdasarkan *pseudocode* Gambar 1, maka secara umum tahapan proses dalam algoritma *Damerau Levenshtein Distance* adalah sebagai berikut [9] :

1. Melakukan proses inisialisasi nilai "a" sebagai ukuran banyaknya huruf dari *string* s dan "b" sebagai jumlah banyaknya karakter dari t. Apabila "a" bernilai = 0 atau banyaknya karakter "b" bernilai = 0, maka jarak akan dikembalikan menggunakan perhitungan seperti pada persamaan (1) berikut :
   jarak edit $= max(n, m)$ ... (1)
   kemudian langsung menuju pada langkah ke 7 (selesai).
2. Membuat sebuah matriks d dengan banyak baris o + 1 dan kolom p + 1.
3. Baris pertama akan diisi dengan nilai 0..a dan pada kolom pertama akan diisi dengan nilai 0..b.
4. Melakukan pemeriksaan terhadap nilai dari masing-masing huruf string s terhadap karakter t.
   Cost = 0 apabila s[o] = t[p]
   Cost = 1 apabila s[o] ≠ t[p]
5. Setiap sel h[i,j] pada setiap baris diisi dengan nilai yang berpatokan pada kaidah persamaan (2) berikut :
   $r[o, p] = min(x, y, z)$ ...(2)
   dimana dari persamaan (2) tersebut diberikan penjelasan sebagai berikut :
   r[o, p] : o & p merupakan pertemuan dalam matriks r antara baris o dengan kolom p.
   x : adalah nilai yang didapat dari penambahan nilai 1 dengan nilai pada posisi sel yang aktif. Nilai ini dapat dirumuskan sebagai :
   $$x = r[o - 1, p] + 1$$
   y : nilai pada posisi sel yang terletak di kiri sel yang aktif dijumlahkan dengan 1 (satu). Nilai ini dirumuskan sebagai :
   $$y = r[o, p - 1] + 1$$
   z : nilai pada posisi sel yang terletak diatas dari posisi sebelah kiri sel yang aktif ditambah cost. Perhitungan ini dituliskan seperti persamaan (3) berikut :
   $z = r[o - 1, p - 1] + cost$ ...(3)
6. Setelah menyelesaikan seluruh langkah perulangan diatas, maka jarak akan ditemukan pada sel yang terletak diposisi kanan pada baris terakhir.
7. Proses berhenti.

B. Algoritma Jaro Winkler Distance

Algoritma *Jaro-Winkler* sering digunakan dalam kegiatan mencari nilai kesamaan antara dua buah string yang dibandingkan. Algoritma ini seringkali disebut sebagai kelompok algoritma *Jaro Distance Metric*. Kegiatan yang paling sering dilakukan dengan menggunakan algoritma ini adalah pendeteksian plagiasi. Secara prinsip, apabila nilai algoritma *Jaro-Winkler Distance* ini bernilai tinggi untuk dua *string* maka dua *string* yang dibandingkan dapat dikatakan memiliki kemiripan. Nilai kemiripan dari hasil pembandingan ini berupa nilai 0 (nol) yang berarti dua *string* yang dibandingkan tidak memiliki kesamaan dan nilai 1 (satu) yang menandakan bahwa dua buah string yang dibandingkan memiliki kesamaan [10]. Menurut Saptono, dkk [11] algoritma *Jaro-Winkler Distance* merupakan algoritma yang baik dan pas untuk digunakan dalam membandingkan string yang jumlah karakternya tidak tidak terlalu panjang. Skor normalnya yakni nol dimana skor ini menandakan string yang dibandingkan tidak sama dan satu yang menandakan sama persis. Menurut Yulianingsih [12], algoritma *Jaro-Winkler* biasanya digunakan dibidang keterkaitan rekaman (duplikat) dan paling cocok digunakan untuk membandingkan *string* yang pendek. Didalam algoritma *Jaro-Winkler Distance* terdapat unsur kompleksitas *runtime* kuadratik dimana kompleksitas ini dinilai sangat efektif untuk string yang tidak terlalu panjang dan mampu menjalankan proses kerja dengan lebih cepat dibandingkan algoritma *edit distance* [11].

Menurut Yulianingsih [12] dasar utama dari algoritma *Jaro Winkler* adalah memiliki kriteria yang terdiri atas proses mencari jumlah panjang kata, mencari jumlah huruf yang sama di dalam dua kata yang dibandingkan dan mencari jumlah perpindahan posisi dari kata tersebut. Perhitungan dari algoritma ini ditunjukkan dalam persamaan (4) :

$$d_j = \frac{1}{3} x \left(\frac{m}{s_1} + \frac{m}{s_2} + \frac{m-t}{m}\right) \quad ...(4)$$

Dari persamaan (4) dapat diberikan keterangan sebagai berikut :

m : banyaknya huruf yang memiliki kesamaan.
s_1 : banyaknya huruf dari kata/*string* pertama
s_2 : banyaknya huruf dari kata/*string* kedua
t : banyaknya perpindahan karakter yang sama pada *string*.

Dalam algoritma ini jarak yang dapat dibenarkan adalah jika jarak tersebut tidak melebihi dari niai dari persamaan (5) berikut :

$$\left(max \frac{s_1 - s_2}{s}\right) - 1 \quad ...(5)$$

Dari persamaan (5) apabila dilakukan pembandingan antara *string* s_1 dan s_2 maka *Jaro-Winkler distance*nya (d_w) dapat dihitung dengan persamaan (6) :

$$d_w = d_j + \left(l\, x\, p\left(1 - d_j\right)\right) \quad ...(6)$$

dimana :

d_j : *Jaro distance* untuk *strings* s_1 dan s_2
l : panjang prefiks umum di awal *string* dengan nilai maksimalnya 4 karakter yakni panjang karakter yang sama sebelum ditemukan.
p : konstanta faktor skala dengan nilai standart nya adalah = 0.1.

C. Pengertian Spelling Checker

Didalam setiap fungsi pengecekan ejaan kata (*spell check*) pada umumnya terdapat beberapa fungsi pendekatan

pencocokan *string* yakni teknik pencocokan *string* yang didasarkan pada kemiripan string yang dibandingkan dari segi penulisannya baik dari jumlah maupun susunan karakternya dan teknik mencari kecocokan antar dua *string* yang dibandingkan berdasarkan pada kemiripan ucapan[13].

Beberapa proses yang dapat dijalankan oleh sebuah *spell checker* antara lain :

1. Naskah yang dicek akan dilakukan pemeriksaan dan di *parsing* seluruh kata yang berada di dalamnya.
2. Hasil *parsing* kata-kata ini akan dibandingkan dengan kamus yang berisi ejaan kata yang benar. Kamus ini digunakan untuk meningkatkan akurasi *spell checker*.
3. Pemanfaatan teknik pencocokan yang sesuai untuk menangani morfologi.

D. . Pengertian String Matching

String dapat diartikan sebagai kumpulan dari satu atau lebih karakter. *String* dan karakter memiliki perbedaan utama dari sisi penulisannya. *String* ditulis menggunakan tanda petik ganda (“ contoh “) , sedangkan karakter ditulis menggunakan tanda petik tunggal (‘ contoh ‘). Menurut Kadir [14], *string* adalah serangkaian karakter yang memiliki ukuran panjang dan dapat dijadikan sebagai tipe data. *String* dirangkai dari unsur-unsur yang memiliki tipe char.

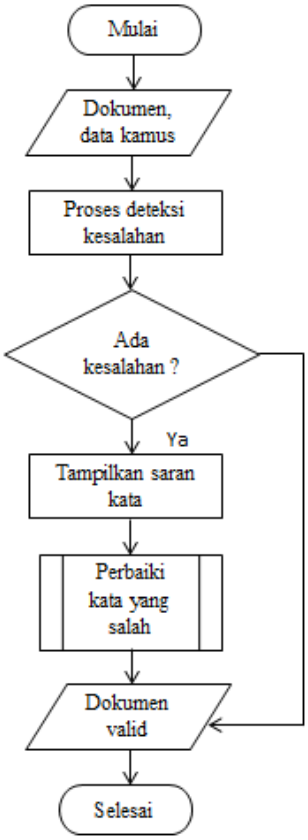
Menurut Sarno [15], *string matching* merupakan rangkaian proses pencarian secara keseluruhan dari kemunculan *query* yang hasil akhirnya dikenal dengan istilah *pattern*. Teknik pencocokan *string* sendiri terbagi menjadi dua kelompok besar yakni pencocokan secara tepat dan dan pencocokan heuristic. Masih menurut Sarno [15], pencocokan secara tepat merupakan pola dalam pencocokan string untuk mendapatkan *pattern* yang dihasilkan dari satu teks, sedangkan pencocokan heuristic merupakan proses pencocokan yang digunakan untuk merelasikan antara dua *string* dengan tempat yang terpisah.

III. HASIL DAN PEMBAHASAN

Pada bagian ini akan dijelaskan tentang alur proses pendeteksian atas kesalahan pengetikan ejaan kata bahasa Inggris. Alur ini menggambarkan tahapan-tahapan proses koreksi kesalahan ejaan dilakukan. Proses ini diawali dengan mendeteksi adanya kesalahan pengetikan sampai dengan menampilkan solusi saran ejaan kata yang tepat. Dari sejumlah solusi saran ejaan kata tersebut sistem akan secara otomatis mengambil kata dengan nilai jarak terkecil untuk dijadikan sebagai ejaan kata yang benar dari sistem. Selain itu, pada bagian ini akan dipaparkan penerapan dari teori algoritma *Damerau Levenshtein Distance* dan algoritma *Jaro Winkler Distance*.

A. Alur Proses Deteksi Kesalahan

Tahapan-tahapan proses deteksi kesalahan pengetikan ejaan kata sampai dengan menampilkan solusi saran kata ditunjukkan seperti dalam Gambar 2 :



Gambar 2. Tahapan proses deteksi kesalahan pengetikkan

Berdasarkan pada gambar 2 diatas, tahapan dimulai dengan menyiapkan dokumen dan kamus kata bahasa Inggris. Setelah semua dokumen dirasa siap, maka selanjutnya dilakukan proses deteksi terhadap ejaan dalam naskah dokumen yang dimasukkan. Proses deteksi ini dilakukan dengan menggunakan kombinasi algoritma *Jaro-Winkler Distance* dan *Damerau Levenshtein Distance*. Pada saat proses deteksi dilakukan dan apabila ditemukan adanya ketidakcocokan antara ejaan dalam dokumen dengan ejaan dalam kamus bahasa Inggris, maka sistem akan menampilkan saran kata yang benar. Untuk menampilkan saran kata ini digunakan algoritma *Jaro-Winkler Distance*.

Proses menampilkan saran kata yang benar dilakukan dengan jalan : 1). mula-mula dokumen yang masuk di cek kesamaannya dengan daftar kata yang ada pada kamus bahasa Inggris. 2). apabila tidak ditemukan perbedaan antara dokumen dengan daftar kata pada kamus bahasa Inggris (ejaan benar), maka algoritma *Jaro-Winkler Distance* tidak melakukan penghitungan jarak diantara keduanya. 3). apabila ternyata ditemukan ketidaksamaan, maka algoritma *Jaro-Winkler Distance* akan bekerja menghitung jarak kedekatan antara ejaan dalam dokumen dengan daftar kata dalam kamus bahasa Inggris yang memiliki kesamaan sesuai dengan banyaknya karakter atau huruf dari ejaan tersebut. Setelah dihitung, maka deretan solusi kata yang benar akan

ditampilkan. Dari pilihan kata yang ditampilkan tersebut, ejaan yang salah dalam naskah diperbaiki berdasarkan pada pilihan solusi kata yang benar. Selanjutnya, apabila sudah tidak ditemukan lagi adanya kesalahan ejaan kata, maka dokumen dianggap valid.

B. Analisis Jaro-Winkler Distance

Algoritma *Jaro-Winkler Distance* prinsip kerjanya adalah melakukan pengecekan dan membandingkan ejaan dari setiap *token* (kata-kata hasil pemecahan kalimat) dengan daftar kata yang terdapat dalam suatu basis data. Apabila hasil pengecekan serta pembandingan tersebut diperoleh kecocokan maka kata yang diproses dianggap benar dan tidak dilakukan penghitungan jarak kedekatannya. Namun sebaliknya, jika ditemukan adanya ketidakcocokan, maka kata yang diproses dianggap salah dan dilakukan penghitungan jarak kedekatannya dengan daftar kata dalam basis data. Nilai jarak kedekatan yang besar menandakan adanya kemiripan antara kata yang dibandingkan, sedangkan nilai jarak kedekatan yang kecil menandakan tidak adanya kemiripan [16].

Indikator yang digunakan dalam penilaian algoritma *Jaro-Winkler* adalah berupa nilai skala awalan (p) yang menggambarkan nilai tertinggi dan panjang awalan (ℓ) dimana indikator ini menggambarkan banyaknya karakter yang memiliki kesamaan dengan kalimat yang akan dibandingkan [5].

Untuk memperjelas penerapan dari algoritma *Jaro Winkler Distance* berikut ini diberikan contoh permasalahan. Diberikan kata (S_1) yang benar dengan kalimat : ORANGE dan kata (S_2) yang salah dengan kalimat : ORAGNE. Dari *string* tersebut diperoleh : m=6; S_1=6 ; S_2=6 dan dapat dilihat bahwa *string* yang tertukar adalah : N dan G sehingga nilai t nya adalah = 1. Dari data tersebut perhitungan nilai jarak *Jaro Winkler* diselesaikan dengan jalan sebagai berikut :

$$d_j = \frac{1}{3}\left(\frac{6}{6}+\frac{6}{6}+\frac{6-1}{6}\right) = 0.944$$

Selanjutnya jika diperhatikan susunan dari kata pertama dan kata kedua dapat diketahui nilai awalan yang sama (l) adalah tiga (3), dan skala nilai p adalah 0.1, maka dapat diperoleh nilai *Jaro Winkler Distance* adalah sebagai berikut :

$d_w = 0.944 + (3 \times 0.1\ (1 - 0.944)) = 0.9608$

Diberikan contoh lain analisis dari algoritma *Jaro Winkler Distance* adalah sebagai berikut : sebagai misal penulis bermaksud untuk mengetik kata : ***Write*** namun salah dengan mengetik kata : ***Wrile***. Oleh sistem dilakukan pengecekan seccara otomatis dan diketahui bahwa kata ***Wrile*** tidak terdapat dalam *database*/kamus bahasa Inggris yang digunakan. Maka dari itu, kata ***Wrile*** akan dikalkulasi dihitung jarak kedekatannya dengan kandidat kata yang tersedia dalam kamus bahasa Inggris. Kandidat saran kata yang diberikan oleh sistem antara lain "***while***", "***wile*** dan "***write***". Penyelesaian dari kasus tersebut adalah sebagai berikut :

Kata : **Wrile** dengan **While**

$$Jaro(S_1S_2) = \frac{1}{3}\left(\frac{m}{|S_1|}+\frac{m}{|S_2|}+\frac{m-t}{|m|}\right)$$

Kata 1 (S_1)	=	Wrile
Kata 2 (S_2)	=	While
Jumlah huruf S_1	=	5
Jumlah huruf S_2	=	5
Jumlah awalan sama (m)	=	1
Jumlah transposisi (t)	=	0

Maka dapat dihitung :

a. Nilai Jaro nya :

$$\frac{1}{3}\left(\frac{1}{5}+\frac{1}{5}+\frac{1-0}{1}\right)$$

= 0.5

b. Nilai jarak kedekatannya :

(Kata1, Kata2) → (Kata1, Kata2) +
(L*p(1 Nilai Jaro(Kata1, Kata2))
Awalan yang sama → 1
Skalar → 0.1
(Kata1, Kata2) → (0.5)+(1*0.1(1-0.5))
(Kata1, Kata2) → (0.5)+(1*0.1-0.05)
(Kata1, Kata2) → (0.5)+(1*0.05)
(Kata1, Kata2) → (0.5)+(0.05) = 0.55

Kata : **Wrile** dengan **Wile**

$$Jaro(S_1S_2) = \frac{1}{3}\left(\frac{m}{|S_1|}+\frac{m}{|S_2|}+\frac{m-t}{|m|}\right)$$

Kata 1 (S_1)	=	Wrile
Kata 1 (S_2)	=	Wile
Jumlah huruf S_1	=	5
Jumlah huruf S_2	=	4
Jumlah awalan sama (m)	=	1
Jumlah transposisi (t)	=	0

Maka dapat dihitung :

a. Nilai Jaro nya :

$$\frac{1}{3}\left(\frac{1}{5}+\frac{1}{4}+\frac{1-0}{1}\right)$$

= 0.48

b. Nilai jarak kedekatannya :

(Kata1, Kata2) → (Kata1, Kata2) +
(L*p(1 Nilai Jaro(Kata1, Kata2))
Awalan yang sama → 1
Skalar → 0.1
(Kata1, Kata2) → (0.48)+(1*0.1(1-0.48))

(Kata1, Kata2) → (0.48)+(1*0.1-0.048)
(Kata1, Kata2) → (0.48)+(1*0.052)
(Kata1, Kata2) → (0.48)+(0.052) = 0.53

Kata : **Wrile** dengan **Write**

$$Jaro(S_1S_2) = \frac{1}{3}\left(\frac{m}{|S_1|} + \frac{m}{|S_2|} + \frac{m-t}{|m|}\right)$$

Kata 1 (S_1)	=	Wrile
Kata 1 (S_2)	=	Write
Jumlah huruf S_1	=	5
Jumlah huruf S_2	=	5
Jumlah awalan sama (m)	=	3
Jumlah transposisi (t)	=	0

Maka dapat dihitung :

a. Nilai Jaro nya :

$$\frac{1}{3}\left(\frac{3}{5} + \frac{3}{5} + \frac{3-0}{3}\right)$$

= 0.73

b. Nilai jarak kedekatannya:

(Kata1, Kata2) → (Kata1, Kata2) +
(L*p(1 Nilai Jaro(Kata1, Kata2))
Awalan yang sama → 3
Skalar → 0.1
(Kata1, Kata2) → (0.73)+(3*0.1(1-0.73))
(Kata1, Kata2) → (0.73)+(3*0.1-0.073)
(Kata1, Kata2) → (0.73)+(3*0.02
(Kata1, Kata2) → (0.73)+(0.081) = 0.811

Dari perhitungan diatas, maka kesimpulan hasil perhitungannya diperlihatkan dalam Tabel I :

TABEL I
HASIL PERHITUNGAN JARO WINKLER DISTANCE

No	Hasil Perhitungan		
	String 1	String 2	Nilai Jaro Winkler
1	Wrile	While	0.55
2	Wrile	Wile	0.53
3	Wrile	Write	0.811

Dari hasil perhitungan yang tampak dalam Tabel I, kata "**wrile**" memiliki tingkat kemiripan paling dekat dengan kata "**Write**" dimana nilai kedekatannya sebesar 0,811, sehingga kata **write** dinilai paling tepat digunakan untuk menggantikan kata **Wrile**.

C. Analisis Damerau Levenshtein Distance

Konsep utama dari jalannya algoritma ini adalah melakukan proses pembandingan terhadap kata-kata yang dibandingkan dengan menitikberatkan pada empat jenis kesalahan yakni :

1. Penyisipan sebuah huruf atau karakter. Sebagai contoh **One** disisipi 1 huruf menjadi **Once**.
2. Penghapusan sebuah huruf atau karakter. Sebagai contoh kata **One** dihapus menjadi **On**.
3. Penggantian sebuah huruf dengan huruf lain. sebagai contoh kata **One** diganti dengan **Ole**.
4. Penukaran huruf yang saling berurutan. Sebagai contoh **One** ditukar hurufnya menjadi **Oen**.

Untuk menghitung jarak dalam algoritma *Damerau Levenshtein Distance* dapat digunakan contoh yang ditunjukkan dalam Tabel II. Diberikan contoh ejaan yang salah sebagai berikut :

S : WHNE → m=4 → Kata yang salah
T : WHEN → n=4. → kata yang benar

Maka jika dilihat dari langkah-langkah penyelesaian algoritma *Damerau Levenshtein Distance* dapat dihitung jaraknya seperti dalam Tabel II berikut :

TABEL II
CONTOH HASIL PERHITUNGAN

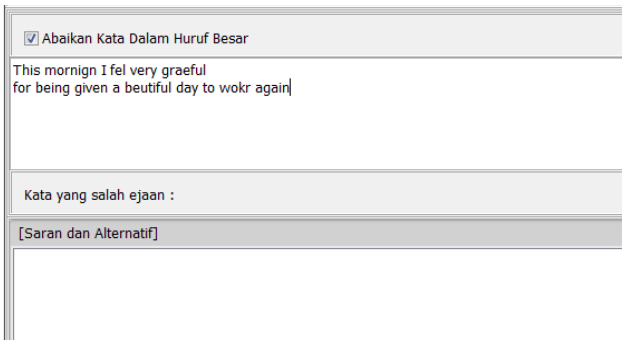
		String Target				
			W	H	E	N
		0	1	2	3	4
String Sumber	W	1	0	1	2	3
	H	2	1	0	1	2
	N	3	2	1	1	1
	E	4	3	2	1	1

Dari Tabel II terlihat beberapa sel yang memuat beberapa karakter yang akan dicari jaraknya. Penempatan data dimulai dari posisi baris 1 dan kolom 1. Sel-sel dalam tabel diisi sesuai dengan konsep atau *pseudocode* dari algoritma *Damerau Levenshtein Distance* yang diperlihatkan dalam Gambar 1.

D. Tampilan Program

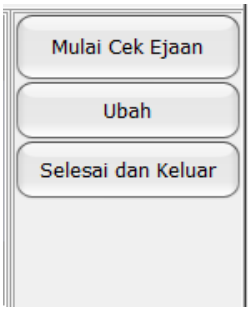
Aplikasi yang dikembangkan ini dapat dipergunakan untuk melakukan deteksi dan perbaikan terhadap kosakata bahasa Inggris yang tidak sesuai. Koreksi dilakukan dengan menampilkan solusi kata yang tepat sehingga mampu untuk melakukan proses : penggantian, penukaran, penyisipan maupun penghapusan karakter dari istilah yang salah. Tampilan antarmuka untuk melakukan deteksi istilah bahasa Inggris diperlihatkan seperti Gambar 3 :

p-ISSN : 2443-2210
e-ISSN : 2443-2229



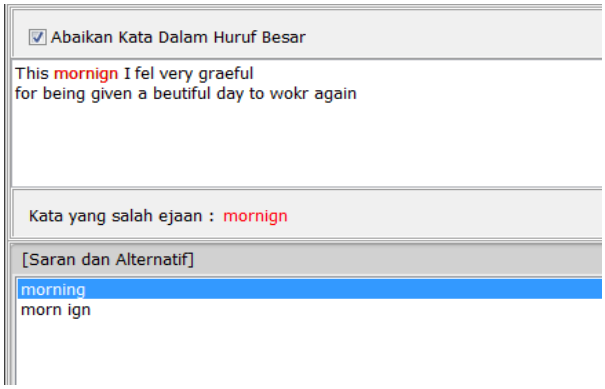
Gambar 3. Form proses deteksi kesalahan pengetikan

Gambar 3 memperlihatkan antarmuka yang berisi teks yang banyak mengandung kesalahan pengetikan. Dalam antarmuka ini disediakan beberapa tombol proses seperti ditunjukkan Gambar 4.



Gambar 4. Tombol untuk melakukan pengecekan ejaan**.**

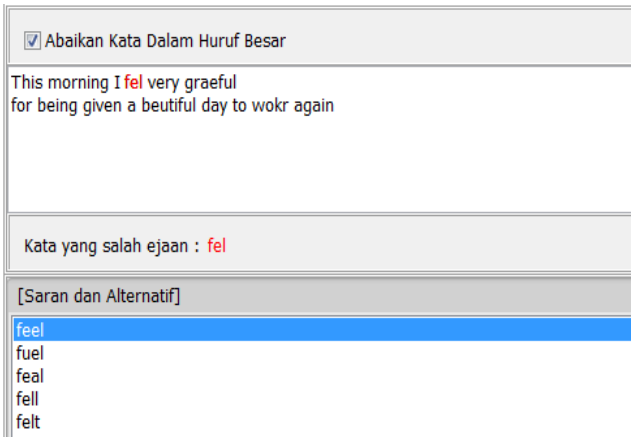
Dari tombol-tombol yang diperlihatkan dalam Gambar 5 untuk mengetahui pada ejaan kata mana saja yang salah, maka pengguna dapat menekan tombol Mulai Cek Ejaan. Hasilnya diperlihatkan pada Gambar 5 :



Gambar 5. Antarmuka proses pengecekan ejaan.

Dari Gambar 5 terlihat apabila terdapat kesalahan pengetikan ejaan, maka secara otomatis sistem akan mendeteksi pada kata ejaan apa yang salah dengan memberikan tanda warna teks merah. Kemudian untuk memudahkan pengguna dalam melakukan koreksi kata yang salah, maka secara otomatis sistem akan menampilkan solusi saran kata yang benar. Saran kata yang di tampilkan sebagai solusi ini berdasarkan pada proses perhitungan algoritma *Jaro-Winkler Distance*.

Untuk melakukan koreksi dan pembetulan, pengguna dapat melakukan klik ganda pada saran kata atau dengan menekan tombol **ubah**. Sebagai contoh kata **mornign** akan diganti dengan **morning.** Hasilnya diperlihatkan pada Gambar 6 :



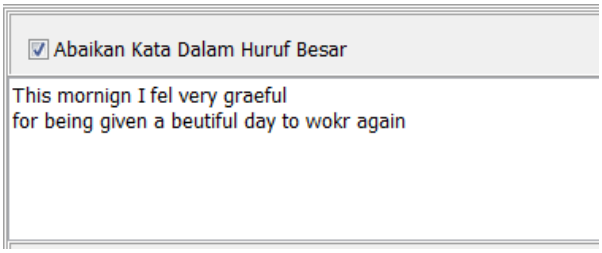
Gambar 6. Antarmuka hasil pengecekan ejaan.

Terlihat dari Gambar 6, kata ***mornign*** telah diganti dengan kata ***morning***. Kemudian jika kesalahan pertama selesai diperbaiki, maka sistem akan melakukan proses pengecekan kembali terhadap seluruh ejaan dalam naskah. Apabila masih ditemukan adanya kesalahan, maka seperti proses awal tadi, sistem akan kembali menandai ejaan yang salah dan memberikan solusi ejaan yang benar. Proses ini akan terus dilakukan sampai tidak ditemukannya kesalahan ejaan atau pengetikan.

Apabila pengguna ingin membatalkan hasil koreksi dan perbaikan ejaan yang salah, dari beberapa tombol yang tersedia seperti Gambar 7, maka pengguna dapat melakukan klik pada tombol batal. Berikut tampilan tombol proses yang disediakan saat proses berlangsung.

Gambar 7. Tombol proses pengecekan ejaan.

Pada saat dilakukan penekanan tombol batal**,** hasil perbaikan yang telah dilakukan akan kembali seperti kondisi awal yang diperlihatkan seperti dalam Gambar 8.



Gambar 8. Hasil pembatalan proses koreksi.

IV. PENGUJIAN

Pada bagian ini akan dilakukan pengujian *unit testing* yakni pengujian untuk melihat apakah algoritma yang diterapkan sudah berjalan sesuai dengan fungsinya, baik fungsi deteksi kesesuaian ejaan maupun fungsi menampilkan solusi saran kata.

A. Tokenisasi

Tokenisasi adalah tahap untuk melakukan pemisahan sebuah kalimat menjadi beberapa kata yang dikenal dengan istilah *token*, sehingga *token* dapat diartikan sebagai kata-kata yang dihasilkan dari pemisahan teks aslinya tanpa mempertimbangkan adanya kerangkapan. Berikut contoh proses tokenisasi yang dikemas dalam bentuk *array*.

Ejaan bahasa Indonesia : Hari Ini Belajar Algoritma
Ejaan bahasa Inggris : Today learn algorithm
Ejaan salah ketik : Today leanr algorhtm
Token : [0]→ Today
[1]→ leanr
[2]→ algorhtm

B. Pengujian Algoritma Damerau Levenshtein Distance

Setelah proses tokenisasi selesai dilakukan, maka hasil tokenisasi diuji dengan algoritma *Damerau Levenshtein Distance* untuk mencari jarak terdekat dari kata-kata yang memiliki kemiripan dalam kamus bahasa Inggris. Berikut disajikan contoh proses pengujiannya.

Ejaan yang salah = *leanr*
Solusi kata = *learn, leans, learner*

Maka dari data tersebut, akan dihitung jarak terdekat dari kata-kata dalam kamus bahasa Inggris yang memiliki kemiripan dengan ejaan yang salah. Pengujian pertama dilakukan terhadap kata ***leanr*** dengan kata ***learn*** seperti diperlihatkan dalam Gambar 9.

		L	E	A	N	R
	0	1	2	3	4	5
L	1	0	1	2	3	4
E	2	1	0	1	2	3
A	3	2	1	0	1	2
R	4	3	2	1	1	0
N	5	4	3	2	0	1

Gambar 9. Hasil pengujian pertama

Dari gambar 9 dapat diberikan penjelasan cara perhitungan sebagai berikut :

1. Isi nilai tiap sel baris per baris
2. Jika huruf pertama yang dibandingkan memiliki kesamaan, maka sel diisi dengan nol (0).
3. Misal : d(1,1). Karena huruf yang dibandingkan sama yakni huruf L, maka sel (baris ke-1,kolom ke-1) diisi dengan 0 (nol).
4. d(1,2) didapat dengan perhitungan berikut :
   =min(nilai baris-1,nilai kolom)+1,(nilai baris,nilai kolom-1)+1,(nilai baris-1,nilai kolom-1)+1
   =min(1-1,2)+1,(1,2-1)+1,(1-1,2-1)+1
   =min(0,2)+1,(1,1)+1,(0,1)+1
5. Selanjutnya kita lihat nilai pada baris ke-0 kolom ke 2. Didapat nilai 2. Kemudian ditambah 1 sehingga menjadi 3. Baris ke-1 kolom ke-1 bernilai 0 + 1 = 1. Baris ke-0 kolom ke-1 bernilai 1+1= 2
6. Dari nilai-nilai yang didapatkan pada nomor 5, maka kita cari nilai paling kecil yakni : min(3,1,2) = 1.
7. Masukkan nilai 1 pada sel (baris ke-1,kolom ke-2)
8. Lakukan proses seperti ini dengan cara yang sama untuk semua sel seperti Gambar 9.

Pengujian kedua dilakukan dengan menguji kata ***leanr*** dengan solusi kata : ***leans.*** Dengan menggunakan cara yang sama seperti pengujian pertama maka didapatkan hasil seperti diperlihatkan pada Gambar 10.

		L	E	A	N	R
	0	1	2	3	4	5
L	1	0	1	2	3	4
E	2	1	0	1	2	3
A	3	2	1	0	1	2
N	4	3	2	1	0	1
S	5	4	3	2	1	1

Gambar 10. Hasil pengujian kedua

Pengujian ketiga dilakukan terhadap kata ***leanr*** dengan kata : ***learner.*** Dengan menggunakan cara perhitungan yang sama didapatkan hasil seperti pada Gambar 11.

		L	E	A	N	R
	0	1	2	3	4	5
L	1	0	1	2	3	4
E	2	1	0	1	2	3
A	3	2	1	0	1	2
R	4	3	2	1	1	0
N	5	4	3	2	2	1
E	6	5	4	3	3	2
R	7	6	5	4	4	3

Gambar 11. Hasil pengujian ketiga

Dari hasil pengujian ejaan kata ***leanr*** maka didapatkan hasil seperti dalam Tabel III.

p-ISSN : 2443-2210
e-ISSN : 2443-2229

TABEL IIII
HASIL PENGUJIAN KATA LEANR

Kata salah	Solusi kata	Jarak
Leanr	Learn	1
	Leans	1
	Learner	3

Berdasarkan data pada Tabel III dapat dilihat bahwa solusi kata : *learn* dan *leans* memiliki nilai jarak yang sama yakni 1. Namun apabila kalimat dalam dokumen tersebut dirangkai dan diartikan dalam bahasa Indonesia, maka kata *learn* menjadi alternatif yang tepat untuk menggantikan kata *leanr*.

Selanjutnya akan dilakukan pengujian kata kedua yang mengalami kesalahan pengetikan yakni kata : ***Algorhtm***. Solusi kata yang tersedia saat kata *algorhtm* di deteksi salah ejaan antara lain : ***algorithm***, ***algometer, algophobia***.

Berikut hasil proses pengujian yang pertama antara kata ***Algorhtm*** dengan kata ***algorithm*** diperlihatkan dalam Gambar 12.

		A	L	G	O	R	H	T	M
	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
L	2	1	0	1	2	3	4	5	6
G	3	2	1	0	1	2	3	4	5
O	4	3	2	1	0	1	2	3	4
R	5	4	3	2	1	0	1	2	3
I	6	5	4	3	2	1	1	2	3
T	7	6	5	4	3	2	2	0	1
H	8	7	6	5	4	3	0	1	1
M	9	8	7	6	5	4	1	1	0

Gambar 12. Hasil pengujian pertama

Pengujian kedua dilakukan dengan menguji kata ***Algorhtm*** dengan solusi kata : ***algometer.*** Dengan menggunakan cara perhitungan yang sama seperti pengujian sebelumnya didapatkan hasil seperti Gambar 13.

		A	L	G	O	R	H	T	M
	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
L	2	1	0	1	2	3	4	5	6
G	3	2	1	0	1	2	3	4	5
O	4	3	2	1	0	1	2	3	4
M	5	4	3	2	1	1	2	3	0
E	6	5	4	3	2	2	2	3	1
T	7	6	5	4	3	3	3	3	2
E	8	7	6	5	4	4	4	4	3
R	9	8	7	6	5	5	5	5	4

Gambar 13. Hasil pengujian kedua

Pengujian ketiga dilakukan terhadap kata ***Algorhtm*** dengan kata kunci yang keluar yakni : ***algophobia***. Hasil pengujian di tunjukkan dalam Gambar 14.

		A	L	G	O	R	H	T	M
	0	1	2	3	4	5	6	7	8
A	1	0	1	2	3	4	5	6	7
L	2	1	0	1	2	3	4	5	6
G	3	2	1	0	1	2	3	4	5
O	4	3	2	1	0	1	2	3	4
P	5	4	3	2	1	1	2	3	4
H	6	5	4	3	2	2	0	1	2
O	7	6	5	4	3	3	1	1	2
B	8	7	6	5	4	4	2	2	2
I	9	8	7	6	5	5	3	3	3
A	10	9	8	7	6	6	4	4	4

Gambar 14. Hasil pengujian ketiga

Dari hasil pengujian ejaan yang salah ***Algorhtm*** maka didapatkan hasil seperti dalam Tabel IV.

TABEL IV
HASIL PENGUJIAN KATA ALGORHTM

Kata salah	Solusi kata	Jarak
Algorhtm	algorithm	0
	algometer	4
	algophobia	4

Berdasarkan data pada Tabel IV dapat dilihat bahwa solusi kata : ***Algorithm*** dianggap paling tepat untuk menggantikan kata ***Algorhtm*** karena memiliki nilai jarak yang paling kecil yakni 0.

C. Pengujian Algoritma Jaro-Winkler Distance

Setelah dilakukan pengujian dengan algoritma *Damerau Levenshtein Distance* selanjutnya akan dilakukan pengujian dengan algoritma *Jaro-Winkler Distance*. Pengujian pertama dilakukan untuk kata ***leanr*** dengan saran kata ***learn***. Proses perhitungannya adalah sebagai berikut :

Pengujian pertama kata ***leanr*** dengan ***learn***.
Kata 1 (S_1) : leanr
Kata 1 (S_2) : learn
Panjang S_1 : 5
Panjang S_2 : 5
Awalan sama : 3
Nilai Jaro di hitung sebagai berikut :

$$\frac{1}{3}\left(\frac{3}{5}+\frac{3}{5}+\frac{3-0}{3}\right)$$

$$\frac{1}{3}\left(\frac{3}{5}+\frac{3}{5}+\frac{3}{3}\right)$$

$$\frac{1}{3}\left(\frac{9}{15}+\frac{9}{15}+\frac{15}{15}\right)$$

$$\frac{1}{3}\left(\frac{33}{15}\right)=0.73$$

Setelah diketahui nilai Jaro nya, selanjutnya dihitung nilai jarak kedekatannya dengan perhitungan berikut :

(0.73)+(1x0.1(1-0.73))
(0.73)+(1x0.1-0.073)
(0.73)+(1x0.027) = 0.75

Pengujian kedua adalah menguji kata ***leanr*** dengan solusi kata ***leans.***

Kata 1 (S_1) : leanr
Kata 1 (S_2) : leans
Panjang S_1 : 5
Panjang S_2 : 5
Awalan sama : 3
Nilai Jaro di hitung sebagai berikut :

$$\frac{1}{3}\left(\frac{3}{5}+\frac{3}{5}+\frac{3-0}{3}\right)$$

$$\frac{1}{3}\left(\frac{3}{5}+\frac{3}{5}+\frac{3}{3}\right)$$

$$\frac{1}{3}\left(\frac{9}{15}+\frac{9}{15}+\frac{15}{15}\right)$$

$$\frac{1}{3}\left(\frac{33}{15}\right) = 0.73$$

Setelah diketahui nilai Jaro nya, selanjutnya dihitung nilai jarak kedekatannya dengan perhitungan berikut :

(0.73)+(1x0.1(1-0.73))
(0.73)+(1x0.1-0.073)
(0.73)+(1x0.027) = 0.75

Pengujian ketiga dilakukan terhada kata ***leanr*** dengan ***learner.***

Kata 1 (S_1) : leanr
Kata 1 (S_2) : learner
Panjang S_1 : 5
Panjang S_2 : 7
Awalan sama : 3
Nilai Jaro di hitung sebagai berikut :

$$\frac{1}{3}\left(\frac{3}{5}+\frac{3}{7}+\frac{3-0}{3}\right)$$

$$\frac{1}{3}\left(\frac{3}{5}+\frac{3}{7}+\frac{3}{3}\right)$$

$$\frac{1}{3}\left(\frac{63}{105}+\frac{45}{105}+\frac{105}{105}\right)$$

$$\frac{1}{3}\left(\frac{213}{105}\right) = 0.67$$

Setelah diketahui nilai Jaro nya, selanjutnya dihitung nilai jarak kedekatannya dengan perhitungan berikut :

(0.67)+(1x0.1(1-0.67))
(0.67)+(1x0.1-0.067)
(0.67)+(1x0.033) = 0.70

Berdasarkan hasil pengujian kata pertama yang salah, maka dapat disimpulkan hasil perhitungan dengan algoritma *Jaro-Winkler Distance* seperti dalam Tabel V.

TABEL V
HASIL PENGUJIAN KATA LEANR

String 1	String 2	Nilai Jaro Winkler
Leanr	Learn	0.73
Leanr	Leans	0.73
Leanr	Learner	0.70

Dari data Tabel V dapat di simpulkan bahwa kata *learn* dan *leans* memiliki nilai *Jaro Winkler* yang tinggi yakni 0.73. Namun jika dilihat dari dokumen aslinya dan apabila diterjemahkan artinya dalam bahasa Indonesia, maka kata *learn* dianggap kata yang lebih tepat sebagai solusi untuk memperbaiki kata *leanr.*

Setelah kata pertama diuji, maka selanjutya dilakukan pengujian terhadap kata kedua yang salah ketik yakni : ***Algorhtm.*** Pengujian pertama dilakukan terhadap kata ***Algorhtm*** dengan kata ***algorithm.***

Kata 1 (S_1) : algorhtm
Kata 1 (S_2) : algorithm
Panjang S_1 : 8
Panjang S_2 : 9
Awalan sama : 5
Nilai Jaro di hitung sebagai berikut :

$$\frac{1}{3}\left(\frac{5}{8}+\frac{5}{9}+\frac{5-0}{5}\right)$$

$$\frac{1}{3}\left(\frac{5}{8}+\frac{5}{9}+\frac{5}{5}\right)$$

$$\frac{1}{3}\left(\frac{225}{360}+\frac{200}{360}+\frac{360}{360}\right)$$

$$\frac{1}{3}\left(\frac{785}{360}\right) = 0.72$$

Setelah diketahui nilai Jaro nya, selanjutnya dihitung nilai jarak kedekatannya dengan perhitungan berikut :

(0.72)+(1x0.1(1-0.72))
(0.72)+(1x0.1-0.072)
(0.72)+(1x0.028) = 0.74

Pengujian kedua adalah menguji kata ***algorhtm*** dengan solusi kata ***algophobia.***

Kata 1 (S_1) : algorhtm
Kata 1 (S_2) : algophobia
Panjang S_1 : 8
Panjang S_2 : 10
Awalan sama : 4

Nilai Jaro di hitung sebagai berikut :

$$\frac{1}{3}\left(\frac{4}{8}+\frac{4}{10}+\frac{4-0}{4}\right)$$

$$\frac{1}{3}\left(\frac{4}{8}+\frac{4}{10}+\frac{4}{4}\right)$$

$$\frac{1}{3}\left(\frac{20}{40}+\frac{16}{40}+\frac{40}{40}\right)$$

$$\frac{1}{3}\left(\frac{76}{120}\right)=0.63$$

Setelah diketahui nilai Jaro nya, selanjutnya dihitung nilai jarak kedekatannya dengan perhitungan berikut :

(0.63)+(1x0.1(1-0.63))
(0.63)+(1x0.1-0.063)
(0.63)+(1x0.037) = 0.66

Pengujian ketiga adalah menguji kata ***algorhtm*** dengan solusi kata ***algometer*.**

Kata 1 (S_1) : algorhtm
Kata 1 (S_2) : algometer
Panjang S_1 : 8
Panjang S_2 : 9
Awalan sama : 4
Nilai Jaro di hitung sebagai berikut :

$$\frac{1}{3}\left(\frac{4}{8}+\frac{4}{9}+\frac{4-0}{4}\right)$$

$$\frac{1}{3}\left(\frac{4}{8}+\frac{4}{9}+\frac{4}{4}\right)$$

$$\frac{1}{3}\left(\frac{36}{72}+\frac{32}{72}+\frac{72}{72}\right)$$

$$\frac{1}{3}\left(\frac{140}{72}\right)=0.65$$

Setelah diketahui nilai Jaro nya, selanjutnya dihitung nilai jarak kedekatannya dengan perhitungan berikut :

(0.65)+(1x0.1(1-0.65))
(0.65)+(1x0.1-0.065)
(0.65)+(1x0.035) = 0.69

Berdasarkan hasil pengujian kata kedua yang salah, maka dapat disimpulkan hasil perhitungan dengan algoritma *Jaro-Winkler Distance* seperti dalam Tabel VI.

TABEL VI
HASIL PENGUJIAN KATA ALGORHTM

String 1	String 2	Nilai Jaro Winkler
Algorhtm	Algorithm	0.74
Algorhtm	Algophobia	0.66
Algorhtm	Algometer	0.69

Dari data Tabel VI dapat di simpulkan bahwa kata *algorithm* memiliki nilai *Jaro Winkler* yang tinggi yakni 0.74, sehingga kata tersebut dianggap tepat untuk memperbaiki kata *algorhtm*.

V. KESIMPULAN

Dari hasil pengujian yang telah dilakukan baik dengan menggunakan algoritma *Damerau Levenshtein Distance* maupun *Jaro-Winkler Distance* maka dapat disimpulkan bahwa penggunaan kedua algoritma tersebut dapat secara optimal membantu mendeteksi adanya ketidaksesuaian ejaan kata bahasa Inggris dalam dokumen dengan sumber data yang ada (kamus). Algoritma *Damerau Levenshtein Distance* mampu melakukan koreksi kata yang salah dan memberikan solusi kata dengan menghitung jarak terdekat (terkecil) antara ejaan dari dokumen dengan ejaan dalam kamus bahasa Inggris yang memiliki kemiripan, sedangkan *Jaro-Winkler Distance* mampu memberikan solusi kata yang tepat berdasarkan pada nilai jarak yang tinggi dari perbandingan antara ejaan kata dalam dokumen dengan kata dalam kamus bahasa Inggris.

DAFTAR PUSTAKA

[1] M. Y. Soleh & A. Purwarianti, "A Non Word Error Spell Checker for Indonesian using Morphologically Analyzer and HMM", *Proceeding of the 2011 International Conference on Electrical Engineering and Informatics*, 2011, p. 1-6.

[2] V. Sagita & M.I. Prasetiyowati, "Studi Perbandingan Implementasi Algoritma BoyerMoore, Turbo Boyer-Moore, dan Tuned Boyer-Moore dalam Pencarian String", *ULTIMATICS*., vol. IV, no. 1, pp. 31-37, Jun. 2013.

[3] P. Santoso, P. Yuliawati, R. Shalahuddin, & I.A.E. Zaeni, "Penghapusan Kolom Dan Baris Pertama Pada Matriks Distance Untuk Optimasi Spell Checker Damerau-Levenshtein Distance", *Sains Aplikasi Komputasi dan Teknologi Informasi*., vol. 2, no. 2, pp. 57-63, Apr. 2020.

[4] E.D. Oktaviyani, S. Christina & D. Ronaldo," Keywords Search Correction Using Damerau Levenshtein Distance Algorithm", *Prosiding Seminar Nasional Teknologi Informasi dan Kedirgantaraan (SENATIK)*, 2019, p. 167-176.

[5] Y. Rochmawati, Y & R. Kusumaningrum, "Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks", *Jurnal Buana Informatika*, vol 7, no 2, pp. 125-134, Apr. 2016.

[6] T.N. Maghfira, I. Cholissodin,& A.W. Widodo, "Deteksi Kesalahan Ejaan dan Penentuan Rekomendasi Koreksi Kata yang Tepat Pada Dokumen Jurnal JTIIK Menggunakan Dictionary Lookup dan Damerau-Levenshtein Distance*", Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*., vol. 1, no. 6, pp. 498-506, Jun. 2017.

[7] U. Sutisna, & J. Adisantoso, "Koreksi Ejaan Query Bahasa Indonesia Menggunakan Algoritma Damerau Levenshtein", *Jurnal Ilmiah Ilmu Komputer*., vol. 15 no. 2, pp. 25-29, Des. 2010.

[8] C. Zhao & S. Sahni, "String correction using the Damerau-Levenshtein distance", *Proceeding IEEE International Conference on Computational Advances in Bio and Medical Sciences (ICCABS 2017): bioinformatics*, 2017, p. 19-46.

[9] A. Pahdi, "Koreksi Ejaan Istilah Komputer Berbasis Kombinasi Algoritma Damerau-Levenshtein dan Algoritma Soundex", *Journal Speed – Sentra Penelitian Engineering dan Edukasi*., vol. 8, no 2,, pp. 1-8, Mei. 2016.

[10] Tinaliah & T. Elizabeth, "Perbandingan Hasil Deteksi Plagiarisme Dokumen dengan Metode JaroWinkler Distance dan Metode Latent

Semantic Analysis", *Jurnal Teknologi dan Sistem Komputer*, vol. 6, no. 1, pp. 7-12, Jan. 2018.

[11] F. Okta'mal, R. Saptono, & M.E. Sulistyo, "Jaro-Winkler Distance Dan Stemming Untuk Deteksi Dini Hama Dan Penyakit Padi", *Prosiding Seminar Nasional Sistem Informasi Indonesia*, 2015, p. 305-312.

[12] Yulianingsih, "Implementasi Algoritma Jaro-Winkler dan Levenstein Distance Dalam Pencarian Data Pada Database", *Jurnal String*, vol. 2, no. 1, pp. 18-27, Agust. 2017.

[13] E.V. Haryanto, "Rancang Bangun Prototype Mesin Pencari String Menggunakan Metode Fuzzy String Matching", *Konferensi Nasional Sistem dan Informatika KNS & I*, 2011, p. 76-82.

[14] A. Kadir, *Algoritma dan Pemrograman Menggunakan Java*, Yogyakarta : Andi, 2012.

[15] R. Sarno, Y. Anistyasari, & R. Fitri, *Semantic Search Pencarian Berdasarkan Konten*, Yogyakarta : Andi, 2012.

[16] A. Prasetyo, W.M. Baihaqi & I.S. Had," Algoritma Jaro-Winkler Distance: Fitur Autocorrect Dan Spelling Suggestion Pada Penulisan Naskah Bahasa Indonesia Di Bms TV", *Jurnal Teknologi Informasi dan Ilmu Komputer (JTIIK)*, vol. 5, no. 4, pp. 435-444, Sept. 2018.