MIND Journal | ISSN (p): 2528-0015 | ISSN (e): 2528-0902 | Vol. 2 | No. 2 | Halaman 1 - 12
DOI : https://doi.org/10.26760/mindjournal Juli 2017

Algoritma Levenshtein *Distance* dalam Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter

Dewi Rosmala, Zulfikar Muhammad Risyad

Jurusan Teknik Informatika, Fakultas Teknologi Industri
Institut Teknologi Nasional Bandung

Email: d_rosmala@itenas.ac.id

ABSTRAK

Twitter digunakan Pemerintah Kota Bandung dalam memperoleh data terkait isu yang terjadi di Kota Bandung yang dilaporkan oleh masyarakat Kota Bandung. Kendala yang dihadapi pada saat melakukan pencarian data isu melalui Twitter adalah adanya perbedaan ejaan kata di dalam tweet pada Twitter dengan kata kunci pada data kategori isu yang ada di Pemerintah Kota Bandung. Sehingga dibutuhkan sebuah algoritma yang mampu mengubah suatu kata menjadi kata lainnya, yaitu Algoritma Levenshtein Distance. Berdasarkan hasil pengujian yang telah dilakukan, Algoritma Levenshtein Distance mampu 100% mengubah kata dengan kesalahan ejaan pada tweet menjadi kata kunci pada kategori isu.Dengan digunakannya Algoritma Levenshtein Distance pada proses pencarian data isu dari Twitter, data isu yang diperoleh Pemerintah Kota Bandung menjadi lebih baik dan akurat.

Kata kunci:*Pencarian Isu, Twitter, Pemerintah Kota Bandung, Algoritma Levenshtein Distance.*

ABSTRACT

Twtter used by The Government of Bandung in obtaining data related issue that happened in Bandung who is submitted by the people of Bandung. Problem that encountered when searching data of issues through Twitter is a difference in spelling the word in a tweet on Twitter with the keyword in category of issue data that exist at The Government of Bandung. So an algorithm that can transform a word into another word is needed, namely Levenshtein Distance Algorithm. Based on the result of testing that has been done, Levenshtein Distance Algorithm 100% capable of changing a word with spelling errors on a tweet into a keyword on category of issue. With the used of Levenshtein Distance Algorithm on the process of searching data issues from Twitter, data issues that obtained by The Government of Bandung become better and more accurate.

Keywords: *Issues Searching, Twitter, The Government of Bandung, Levenshtein Distance Algorithm.*

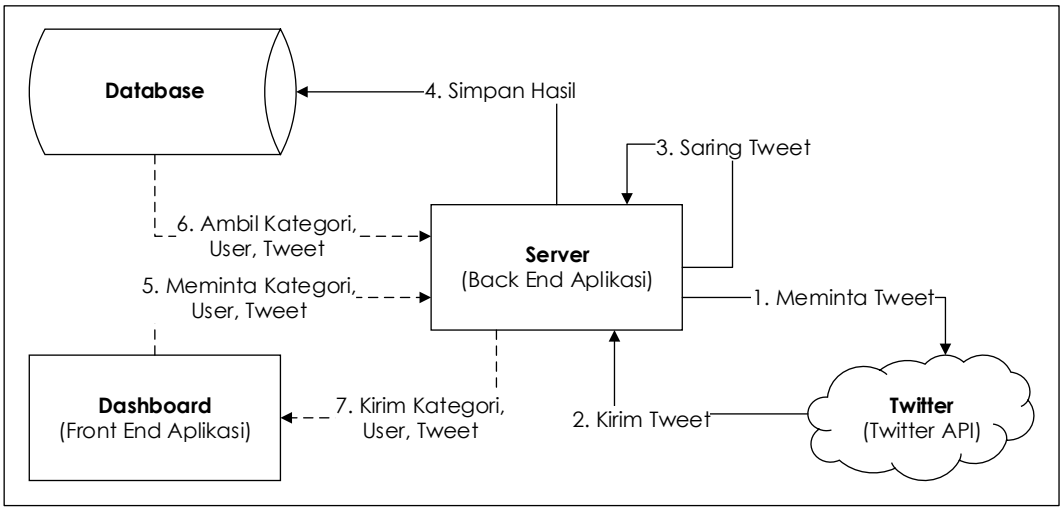
1. LATAR BELAKANG

Proses kerja pemerintahan di Indonesia khususnya pemerintahan di Kota Bandung saat ini menggunakan media sosial sebagai salah satu media pembantu pemerintahan dalam menanggapi isu-isu yang terjadi. Terdapat kendala yang dihadapi Pemerintah Kota Bandung dengan penggunaan Twitter sebagai media pembantu pemerintahan, yaitu kendala pada penulisan ejaan kata yang berbeda dengan kata yang dimaksud pada *tweet* yang dikirimkan oleh pengguna Twitter. Perbedaan ejaan kata pada Twitter tersebut disebabkan oleh beberapa hal, yaitu pengguna menggunakan bahasa yang tidak formal dalam menyampaikan informasi, dan adanya batasan jumlah karakter dalam sebuah *tweet* yang dapat digunakan oleh pengguna Twitter. Oleh karena itu dibutuhkan sebuah algoritma yang dapat mengubah suatu kata menjadi kata lainnya. Ada beberapa algoritma yang dapat melakukan hal tersebut, salah satunya Algoritma Levenshtein *Distance*. Proses kerja Algoritma Levenshtein *Distance* membutuhkan dua buah *string*, kemudian dilakukan perhitungan operasi minimum yang dibutuhkan untuk mengubah satu *string* ke *string* lainnya. Dalam permasalahan ini Algoritma Levenshtein *Distance* digunakan dalam mengubah ejaan kata pada *tweet* di Twitter menjadi kata kunciyang dikategorikan Pemerintah Kota Bandung ke dalam kategori isu.

Algoritma Levenshtein *Distance* diterapkan pada Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter untuk mengubah kata yang memiliki kesalahan ejaan pada *tweet* menjadi kata kunci pada kategori isu dalam *database*. Tujuan dilakukannya penelitian ini adalahmenerapkan Algoritma Levenshtein *Distance* pada aplikasi pencarian data *tweet* mengenai isu di Kota Bandung yang dikirimkan oleh pengguna Twitter. Penelitian dibatasi pada aplikasi yang dibangun, yaitu *tweet* yang masuk ke dalam aplikasi adalah *tweet* yang dialamatkan pada akun Twitter infobdg, dan dinas-dinas Pemerintah Kota Bandung dengan tidak mempedulikan aspek semantik pada isi *tweet*. Jarak Levenshtein *Distance* pada pengubahan kata *tweet* menjadi kata kunci dibatasi dengan jarak 0 (nol) sampai 3 (tiga).

2. METODOLOGI PENELITIAN

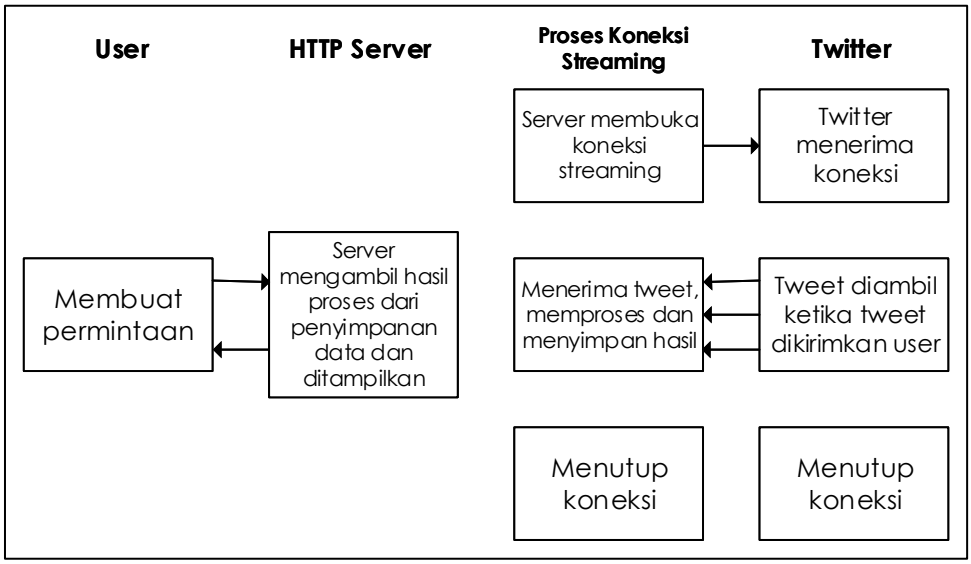
Alur kerja sistem Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter yang dibangun dibagi menjadi 2 (dua) bagian, yaitu bagian *backend* dan bagian *frontend*. Bagian *back end* (tahap 1 sampai 4) bekerja secara terus menerus dimulai dari dijalankannya aplikasi tanpa dibutuhkan interaksi dengan pengguna aplikasi (digambarkan dengan garis lurus). Sedangkan bagian *front end* (tahap 5 sampai 7) hanya bekerja ketika ada interaksi dari pengguna aplikasi (digambarkan dengan garis putus-putus). Pada Gambar 1 dapat diketahui bagaimana sistem Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter bekerja.



Gambar 1. Alur Kerja Sistem

Alur kerja sistem dibagi menjadi 4 (empat) bagian, yaitu *Dashboard*, Server, Twitter, dan *Database*. Bagian *dashboard* atau *frontend* aplikasi menampilkan hasil penyaringan *tweet* menggunakan Algoritma Levenshtein *Distance*yang dilakukan di bagian server ke dalam sebuah tampilan antar muka aplikasi yang dapat dipahami oleh pengguna aplikasi. Bagian server atau *backend* aplikasi menyimpan perintah-perintah untuk menjalankan fungsi permintaan *tweet* dari Twitter dan penyaringan *tweet* menggunakan Algoritma Levenshtein *Distance*. Twitter sebagai aplikasi sosial media yang menyimpan informasi pelaporan isu yang dihubungkan dengan Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter melalui *ApplicationProgrammingInterface* (API). Data kategori yang berisikan nama kategori dan kata kunci, data *tweet* yang berisikan informasi *tweet* hasil penyaringan, dan data *user*yang berisikan informasi *login* disimpan di bagian *database*.

Alur kerja sistem dimulai dengan hubungan antara server dengan Twitter, yaitu permintaan *tweet* berdasarkan kriteria tujuan pengiriman *tweet* dari server kepada Twitter menggunakan Twitter API. Terdapat 2 (dua) metode pengambilan *tweet* yang disediakan Twitter API, yaitu *Streaming* API dan *Rest* API. Metode Streaming API digunakan pada sistem Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter dikarenakan aplikasi membutuhkan data *tweet* yang dikirimkan secara *realtime*. Sedangkan *Rest* API cenderung digunakan oleh *developer* ketika membangun aplikasi yang tidak membutuhkan *tweet* secara *realtime*. Alur kerja pengambilan *tweet* menggunakan *Streaming* API digambarkan pada Gambar 2.

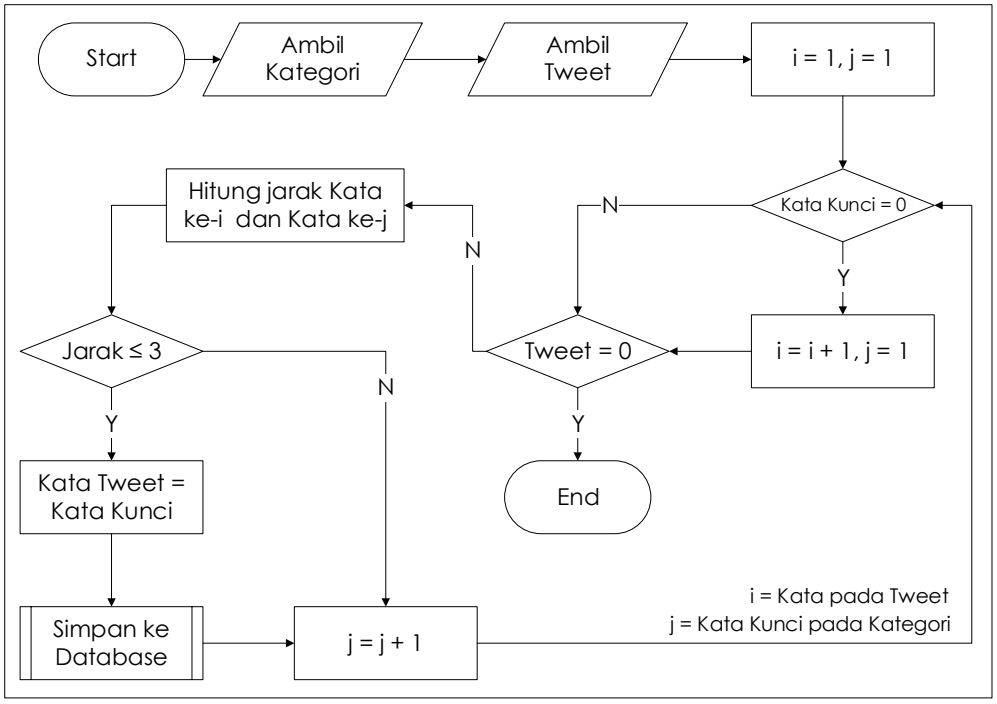


Gambar 2. Alur Kerja *Streaming* API (Sumber: https://dev.twitter.com/streaming/overview)

Proses pengambilan *tweet* menggunakan *Streaming* API dimulai dari pembukaan koneksi *streaming* antara Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter dengan aplikasi Twitter menggunakan OAuth sebagai penyedia otorisasi akses yang digunakan Twitter ketika aplikasi pihak ketiga membutuhkan sambungan ke API. Kemudian*User* atau bagian server Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter melakukan permintaan kepada HTTP Server Twitter. Permintaan tersebut berisikan perintah permintaan *tweet* menggunakan *publicstreams* dengan parameter *replies*, karena *tweet* yang diambil adalah *tweet* dengan kriteria tujuan pengiriman *tweet*. Sehingga isi dari permintaan dari server Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter kepada HTTP Server Twitter adalah GET user </streaming/reference/get/user>.

Proses yang terdapat pada koneksi *streaming* selanjutnya adalah menerima seluruh *tweet* yang dikirimkan oleh pengguna Twitter, lalu dilakukan proses penyaringan berdasarkan kriteria tujuan *tweet*, kemudian *tweet* yang sesuai dengan kriteria tersebut disimpan di HTTP Server Twitter dan ditampilkan sesuai dengan format tampilan yang diminta server Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter sebelum akhirnya dikirimkan. Seluruh proses kerja *Streaming* API tersebut terus berulang sampai Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter menutup koneksi dengan aplikasi Twitter.

Tweet hasil yang dikirimkan Twitter ke server Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter kemudian disaring menggunakan Algoritma Levenshtein *Distance*.Proses penyaringan tersebut adalah perhitungan jarak antara kata pada isi *tweet* dengan kata kunci pada *database*. Proses perhitungan jarak Levenshtein *Distance* tersebut digambarkan dengan diagram alir (*flowchart*) pada Gambar 3.



Gambar 3. *Flowchart* Algoritma Levenshtein *Distance*

Perhitungan jarak Levenshtein *Distance* dilakukan pada setiap kata pada isi *tweet* dengan seluruh kata kunci kategori isu pada *database*. Dengan asumsi kata kunci kategori isu pada *database* dan *tweet* pada server telah tersedia, perhitungan jarak Levenshtein *Distance* dimulai pada kata ke-1 pada *tweet* dengan kata kunci ke-1 pada kategori. Apabila setelah dilakukan perhitungan nilai jarak Levenshtein *Distance* diketahui bernilai 0 (nol) sampai 3 (tiga), maka kata ke-1 pada *tweet* diubah menjadi kata ke-1 pada kategori dan *tweet* sebelum dan setelah pengubahan tersebut disimpan ke *database* beserta informasi kategori dari kata kunci ke-1 tersebut. Tetapi apabila hasil perhitungan nilai jarak Levenshtein *Distance* bernilai lebih dari 3 (tiga), maka perhitungan dilanjutkan pada kata ke-1 pada *tweet* dengan kata kunci ke-2 pada kategori. Proses perhitungan berulang sampai seluruh kata pada *tweet* dihitung jarak Levenshtein *Distance*-nya dengan seluruh kata kunci kategori isu pada *database*. Nilai jarak Levenshtein *Distance* antara 2 (dua) *string* adalah jumlah operasi yang dibutuhkan dalam mengubah satu *string* ke *string* lainnya. Operasi yang digunakan Algoritma Levenshtein *Distance* adalah operasi penghapusan, penyisipan, dan penukaran. Setiap operasi yang digunakan dalam mengubah satu *string* ke *string* lain tersebut bernilai 1 (satu), tetapi apabila tidak dibutuhkan pengubahan maka operasi bernilai 0 (nol).

Setelah sistem menyaring *tweet* dengan menggunakan Algoritma Levenshtein *Distance*, selanjutnya sistem hanya akan bekerja ketika ada interaksi dari pengguna aplikasi terhadap aplikasi melalui bagian *dashboard*. Ketika ada interaksi pengguna aplikasi terhadap bagian dashboard aplikasi, bagian dashboard meminta data ke bagian server berupa data kategori, data tweet, dan data user. Kemudian server mengambil data permintaan tersebut dari database dan mengirimkannya ke dashboard. Data yang ditampilkan pada halaman home dashboard adalah data grafik jumlah tweet per kategori isu, pada halaman kategori adalah nama kategori dan kata kunci, pada halaman tweet adalah kategori tweet, tweet asli sebelum diubah Algoritma Levenshtein Distance, tweet setelah diubah Algoritma Levenshtein Distance, username Twitter pengirim tweet, dan waktu pengriman tweet, pada halaman user adalah Nomor Induk Pegawai (NIP), nama, *username* dan *password login*, dan hak akses pengguna aplikasi.

3. PEMBAHASAN DAN PENGUJIAN

Fungsi yang diuji pada Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter adalah fungsi pengambilan *tweet* dari Twitter dan fungsi penyaringan *tweet* menggunakan Algoritma Levenshtein *Distance*. Pengujian pada fungsi penyaringan *tweet* dilakukan agar dapat diketahui tingkat akurasi pengambilan *tweet* dari Twitter terhadap kriteria yang diberikan. Pengujian terhadap fungsi Algoritma Levenshtein *Distance* dilakukan oleh pihak *developer* (uji *alpha*), baik pengujian pada sistem maupun pengujian secara manual. Fungsi Algoritma Levenshtein *Distance* diuji agar dapat diketahui tingkat akurasi perubahan kata pada *tweet* terhadap kata kunci kategori isu pada *database*. Selanjutnya dapat diketahui kesimpulan dari hasil pengujian seluruh fungsi yang diterapkan pada Aplikasi Pencarian Kata Isu di Kota Bandung pada Twitter.

3.1. Fungsi Pengambilan *Tweet*

Pengujian fungsi pengambilan *tweet* dilakukan untuk mengetahui apakah *tweet*-*tweet* yang dimasukkan ke dalam *database* hanya *tweet* yang ditujukan pada akun Twitter yang telah ditentukan. Kriteria tujuan pengiriman *tweet* yang diambil adalan *tweet* yang dikirim kepada akun Twitter @infobdg dan akun Twitter dinas-dinas Pemerintah Kota Bandung seperti @PemkotBandung (Pemerintah Kota Bandung), @dishub_kotabdg (Dinas Perhubungan Kota Bandung), @DPUKotaBandung (Dinas Perkerjaan Umum Kota Bandung), @distarcipbdg (Dinas Tata Ruang dan Cipta Karya Kota Bandung), @diskar_bdg (Dinas Kebakaran dan Penanggulangan Bencana Kota Bandung), @Dinsos_BDG (Dinas Sosial Kota Bandung), @Satpolppbdg (Satuan Polisi Pamong Praja Kota Bandung), dan @PDKEBERSIHAN (Perusahaan Daerah Kebersihan Kota Bandung).

Setelah fungsi pengambilan *tweet* tersebut diimplementasikan ke dalam sistem, fungsi pengambilan *tweet* diuji dengan tujuan untuk melihat berhasil atau tidaknya penerapan fungsi pengambilan *tweet* pada sistem. Pengujian dilakukan dengan uji *blackbox* dengan informasi aktor yang melakukan pengujian, kondisi awal sebelum dilakukan pengujian, kondisi terakhir yang diharapkan, skenario pengujian, kondisi akhir yang didapatkan, hasil pengujian, dan kesimpulan yang disimpulkan setelah dilakukannya pengujian. Uji *blackbox* pengujian fungsi pengambilan *tweet* dapat diketahui pada Tabel 1.

Tabel 1. Pengujian Fungsi Pengambilan *Tweet*

Nama Fungsi	Pengambilan *Tweet*
Tujuan	Untuk melihat kesesuaian kriteria *tweet*dengan *tweet* dimasukkan yang ke dalam *database*.
Aktor	Admin atau *user*.
Kondisi Awal	Halaman *login*.
Kondisi yang Diharapkan	Daftar *tweet* yang terdapat di *database* sesuai dengan kriteria *tweet*.
Skenario	1) Aktor melakukan *login* ke dalam aplikasi. 2) Aktor berada di halaman*homedashboard*. 3) Aktor menekan tombol menu dan memilih menu *tweet*.
Kondisi Akhir yang Didapatkan	Aktor masuk ke halaman *tweet* dan daftar entri *tweet* yang ada adalah *tweet* yang mengandung *username* akun Twitter yang ditentukan.
Hasil	Berdasarkan kondisi akhir yang diharapkan, *tweet* yang terdapat di *database*sesuai dengan kriteria tujuan *tweet*.
Kesimpulan	*Tweet* yang diambil sesuai dengan kriteria tujuan *tweet*.

3.2. Fungsi Algoritma Levenshtein *Distance*

Matriks 2 (dua) dimensi digunakan dalam perhitungan nilai jarak Levenshtein *Distance*. Isian nilai pada matriks tersebut adalah jumlah operasi penghapusan, penyisipan dan penukaran yang dibutuhkan dalam mengubah *string* sumber ke *string* target. Rumus operasi penghapusan, penyisipan, dan penukaran karakter yang digunakan untuk mengisi nilai matriks adalah sebagai berikut:

$$D(s,t) = \min D(s-1,t) + 1 \text{ (Penghapusan)} \quad (1)$$

$$D(s,t) = min\, D(s,t-1) + 1 \text{ } \textit{(Penyisipan)} \quad (2)$$

$$D(s,t) = min\, D(s-1,t-1) + 1, sj \neq ti \text{ } \textit{(Penukaran)} \quad (3)$$

$$D(s,t) = \min D(s-1,t-1), sj = ti \text{(Tidak ada perubahan)} \quad (4)$$

(Sumber: https://web.stanford.edu/class/cs124/lec/med.pdf)

s= *String* Sumber
t = *String* Target
D = Jarak Levenshtein *Distance*
$s(j)$ = Karakter *String* Sumber ke- j
$t(i)$ = Karakter *String* Target ke- i

Fungsi Algoritma Levenshtein *Distance* diuji pada kasus *tweet* berisi "@PemkotBandung bandung mcet" sebagai *string* sumber dan kata kunci banjir dan kebanjiran yang terdapat pada kategori isu kebanjiran, kata kunci kebakaran yang terdapat pada kategori isu kebakaran, kata kunci kecelakaan dan tabrakan yang terdapat pada kategori isu kecelakaan, kata kunci kemacetan dan macet yang terdapat pada kategori isu kemacetan, kata kunci pencurian dan perampokan yang terdapat pada kategori isu kriminal, kata kunci pelanggaran dan melanggar yang terdapat pada kategori isu pelanggaran, dan kata kunci pengemis yang terdapat pada kategori isu pengemis sebagai *string* target. Berdasarkan contoh kasus tersebut, setiap kata pada *tweet* yaitu "bandung" dan "mcet" dihitung jaraknya dengan seluruh kata kunci. Sedangkan kata "@PemkotBandung" tidak dihitung karena kata tersebut tidak diidentifikasi sebagai isi dari *tweet* oleh Twitter, melainkan *mention*. Langkah awal Algoritma Levenshtein *Distance* adalah menghitung jarak *string* sumber pertama "bandung" dengan *string* target pertama yaitu "banjir". Perhitungan matriks dimulai dengan inisiasi urutan karakter pada masing-masing *string* seperti digambarkan pada Gambar 4.

		***String* Target**						
			b	**a**	**n**	**j**	**i**	**r**
		0	1	2	3	4	5	6
***String* Sumber**	**b**	1						
	a	2						
	n	3						
	d	4						
	u	5						
	n	6						
	g	7						

Gambar 4. Inisiasi Urutan Karakter

Pada Gambar 4 diketahui bahwa *string* sumber "bandung" memiliki 7 (tujuh) karakter dan *string* target "banjir" memiliki 6 (enam) karakter. Selanjutnya karakter ke-1 pada masing-masing *string* dibandingkan dan diketahui bahwa isi karakter ke-1 pada masing-masing *string* sama, maka nilai matriks yang diberikan sesuai dengan Persamaan (4) yaitu $D(1,1) =$

$D(1-1,1-1), sj = ti$. Jadi nilai matriks yang diberikan pada $D(1,1) = D(0,0)$ yang bernilai 0. Kemudian nilai matriks pada $D(1,1)$ diisi seperti yang digambarkan pada Gambar 5.

		***String* Target**						
			b	**a**	**n**	**j**	**i**	**r**
		0	1	2	3	4	5	6
***String* Sumber**	**b**	1	0					
	a	2						
	n	3						
	d	4						
	u	5						
	n	6						
	g	7						

Gambar 5. Jarak D(1,1)

Selanjutnya perhitungan jarak dilakukan pada karakter ke-1 *string* sumber dengan karakter ke-2 pada *string* target dan diketahui bahwa dibutuhkan operasi penyisipan karakter "a" pada *string* sumber, maka nilai yang diberikan sesuai dengan rumus operasi penyisipan pada Rumus $2D(1,2) = D(1,2-1) + 1$. Jadi nilai yang diberikan pada $D(1,2) = D(1,1) + 1$ yang bernilai $D(1,2) = 0 + 1 = 1$. Kemudian nilai matriks pada $D(1,2)$ diisi dengan nilai 1 seperti yang digambarkan pada Gambar 6.

		***String* Target**						
			b	**a**	**n**	**j**	**i**	**r**
		0	1	2	3	4	5	6
***String* Sumber**	**b**	1	0	1				
	a	2						
	n	3						
	d	4						
	u	5						
	n	6						
	g	7						

Gambar 6. Jarak D(1,2)

Selanjutnya perhitungan jarak dilakukan pada karakter ke-1 *string* sumber dengan *string* target sampai dengan karakter ke-3 dan diketahui bahwa dibutuhkan operasi penyisipan karakter "a" dan "n" pada *string* sumber, maka nilai yang diberikan sesuai dengan rumus operasi penyisipan pada Rumus $2D(1,3) = D(1,3-1) + 1$. Jadi nilai yang diberikan pada $D(1,3) = D(1,2) + 1$ yang bernilai $D(1,3) = 1 + 1 = 2$. Kemudian nilai matriks pada $D(1,3)$ diisi dengan nilai 2 seperti yang digambarkan pada Gambar 7.

		***String* Target**						
			b	**a**	**n**	**j**	**i**	**r**
		0	1	2	3	4	5	6
***String* Sumber**	**b**	1	0	1	2			
	a	2						
	n	3						
	d	4						
	u	5						
	n	6						
	g	7						

Gambar 7. Jarak D(1,3)

Selanjutnya perhitungan jarak dilakukan pada karakter ke-1 *string* sumber dengan *string* target sampai dengan karakter ke-4 dan diketahui bahwa dibutuhkan operasi penyisipan karakter "a", "n", dan "j" pada *string* sumber, maka nilai yang diberikan sesuai dengan rumus operasi penyisipan pada Persamaan (2) $D(1,4) = D(1,4-1) + 1$. Jadi nilai yang diberikan pada $D(1,4) = D(1,3) + 1$ yang bernilai $D(1,3) = 2 + 1 = 3$. Kemudian nilai matriks pada $D(1,4)$ diisi dengan nilai 3 seperti yang digambarkan pada Gambar 8.

		***String* Target**						
			b	**a**	**n**	**j**	**i**	**r**
		0	1	2	3	4	5	6
***String* Sumber**	**b**	1	0	1	2	3		
	a	2						
	n	3						
	d	4						
	u	5						
	n	6						
	g	7						

Gambar 8. Jarak D(1,4)

Selanjutnya perhitungan jarak dilakukan pada karakter ke-1 *string* sumber dengan *string* target sampai dengan karakter ke-5 dan diketahui bahwa dibutuhkan operasi penyisipan karakter "a", "n", "j", dan "i" pada *string* sumber, maka nilai yang diberikan sesuai dengan rumus operasi penyisipan pada Persamaan (2) $D(1,5) = D(1,5-1) + 1$. Jadi nilai yang diberikan pada $D(1,5) = D(1,4) + 1$ yang bernilai $D(1,5) = 3 + 1 = 4$. Kemudian nilai matriks pada $D(1,5)$ diisi dengan nilai 4 seperti yang digambarkan pada Gambar 9.

String Sumber		*String* Target						
			b	**a**	**n**	**j**	**i**	**r**
		0	1	2	3	4	5	6
	b	1	0	1	2	3	4	
	a	2						
	n	3						
	d	4						
	u	5						
	n	6						
	g	7						

Gambar 9. Jarak D(1,4)

Selanjutnya perhitungan jarak dilakukan pada karakter ke-1 *string* sumber dengan *string* target sampai dengan karakter ke-6 dan diketahui bahwa dibutuhkan operasi penyisipan karakter "a", "n", "j", "i", dan "r" pada *string* sumber, maka nilai yang diberikan sesuai dengan rumus operasi penyisipan pada Persamaan (2) $D(1,6) = D(1,6-1) + 1$. Jadi nilai yang diberikan pada $D(1,6) = D(1,5) + 1$ yang bernilai $D(1,6) = 4 + 1 = 5$. Kemudian nilai matriks pada $D(1,6)$ diisi dengan nilai 5 seperti yang digambarkan pada Gambar 10.

String Sumber		*String* Target						
			b	**a**	**n**	**j**	**i**	**r**
		0	1	2	3	4	5	6
	b	1	0	1	2	3	4	5
	a	2						
	n	3						
	d	4						
	u	5						
	n	6						
	g	7						

Gambar 10. Jarak D(1,4)

Perhitungan jarak Levenshtein *Distance* selanjutnya berjalan sampai semua nilai pada matriks terisi. Jarak Levenshtein *Distance* adalah nilai yang terdapat di bawah-kanan matriks, dan pada kasus *string* sumber "bandung" dan *string* target "banjir" berada di $D(7,6)$. Setelah dilakukan seluruh perhitungan matriks diketahui hasil dari perhitungan jarak antara *string* sumber "bandung" dan *string* target "banjir" adalah 4 (empat) seperti yang digambarkan pada matriks Gambar 11.

		***String* Target**						
			b	**a**	**n**	**j**	**i**	**r**
***String* Sumber**		0	1	2	3	4	5	6
	b	1	0	1	2	3	4	5
	a	2	1	0	1	2	3	4
	n	3	2	1	0	1	2	3
	d	4	3	2	1	1	2	3
	u	5	4	3	2	2	2	3
	n	6	5	4	3	3	3	3
	g	7	6	5	4	4	4	**4**

Gambar 11. Jarak D(7,6)

Karena batasan masalah pada penelitian ini adalah jarak Levenshtein *Distance* yang digunakan dalam mengubah kata pada *tweet* menjadi kata kunci bernilai 0 (nol) sampai 3 (tiga), maka kata "bandung" tidak diubah menjadi "banjir". Selanjutnya perhitungan dilakukan pada kata kunci kedua sampai dengan kata kunci terakhir sebagai *string* target, sehingga diketahui masing-masing jarak Levenshtein *Distance*-nya seperti pada Tabel 2.

Tabel 2. Jarak Levenshtein *Distance* "bandung" dan Seluruh *String* Target

***String* Sumber**	**Kategori**	***String* Target**	**Jarak**
bandung	Kebanjiran	banjir	4
		kebanjiran	7
	Kebakaran	kebakaran	7
	Kecelakaan	kecelakaan	9
		tabrakan	7
	Kemacetan	kemacetan	8
		macet	6
	Kriminal	pencurian	7
		perampokan	9
	Pelanggaran	pelanggaran	9
		melanggar	7
	Pengemis	pengemis	7

Berdasarkan Tabel 2 diketahui bahwa jarak Levenshtein *Distance* antara *string* sumber "bandung" dengan seluruh *string* target tidak ada yang memiliki jarak 0 (nol) sampai 3 (tiga), maka *string* sumber "bandung" diabaikan dan perhitungan jarak Levenshtein *Distance* berlanjut pada *string* sumber kedua, yaitu "mcet". Perhitungan jarak Levenshtein *Distance* antara *string* sumber "mcet" dengan seluruh *string* target sebagai kata kunci pada kategori isu dapat diketahui pada Tabel 3.

Tabel 3. Jarak Levenshtein *Distance* "mcet" dan Seluruh *String* Target

***String* Sumber**	**Kategori**	***String* Target**	**Jarak**
mcet	Kebanjiran	banjir	6
		kebanjiran	10
	Kebakaran	kebakaran	9
	Kecelakaan	kecelakaan	8
		tabrakan	8
	Kemacetan	kemacetan	5
		macet	1
	Kriminal	pencurian	8
		perampokan	9
	Pelanggaran	pelanggaran	11
		melanggar	8
	Pengemis	pengemis	7

Berdasarkan Tabel 3 diketahui bahwa jarak Levenshtein *Distance* antara *string* sumber "mcet" dengan *string* target "macet" memiliki nilai 1 (satu), maka kemudian *string* sumber "mcet" diubah menjadi *string* target "macet" dan *tweet* "@PemkotBandung bandung mcet" dimasukkan ke dalam kategori "kemacetan".

4. KESIMPULAN

Dari hasil implementasi dan pengujian fungsi yang telah dilakukan, dapat disimpulkan bahwa penggunaan Algoritma Levenshtein *Distance* dalam Aplikasi Pencarian Kata Isu di Kota Bandung Pada Twitter mampu mengubah *tweet* pelaporan isu yang ditujukan pada akun Twitter infobdg dan dinas-dinas Pemerintah Kota Bandung yang mengandung kata dengan kesalahan ejaan menjadi kata kunci yang kemudian dimasukkan ke dalam daftar kategori isu yang terdapat di Pemerintah Kota Bandung dengan tingkat akurasi 100%. Dengan digunakannya Algoritma Levenshtein *Distance* dapat memberikan akurasi data yang lebih baik pada hasil keluaran aplikasi, sehingga dapat digunakan Pemerintah Kota Bandung dalam memperoleh data isu-isu yang dilaporkan warga kepada akun Twitter infobdg dan dinas Pemerintah Kota Bandung.

DAFTAR PUSTAKA

[1] Rochmawati, Yeny. 2015. "Studi Perbandingan Algoritma Pencarian *String*dalam Metode *ApproximateStringMatching*untuk Identifikasi Kesalahan Pengetikan Teks". Universitas Diponegoro, Semarang.

[2] Nangili, S. 2015. "Pengujian Algoritma Levenshtein *Distance* Dan Algoritma *TermFrequencyInverseDocumentFrequency* (TF-IDF) Untuk Penilaian Jawaban Esai". Universitas Negeri Gorontalo, Gorontalo.

[3] Primadani, Yuli. 2014. "Simulasi Algoritma Levenshtein *Distance* Untuk Fitur *Autocomplete* Pada Aplikasi Katalog Perspustakaan". Universitas Sumatera Utara, Medan.

[4] Adriyani, N.M.M. 2012. "Implementasi Algoritma Levenshtein *Distance* dan Metode Empiris untuk Menampilkan Saran Perbaikan Kesalahan Pengetikan Dokumen Berbahasa Indonesia". Universitas Udayana, Badung.

[5] Upadhyay, Abhishanga. 2014. "*MiningDataFrom* Twitter". *UniversityofSouthern* California, Los Angeles

[6] Makice, Kevin. 2009. Twitter API: *Up and Running: Learn How to Build Applications with the* Twitter API. Sebastopol: O'Reilly Media.

[7] Zaki, Mohammed J.,Meira Jr., Wagner. 2014. *DataMiningandAnalysis*: *FundamentalConceptsandAlgorithms*. Cambridge: Cambridge *UniversityPress*.