PROGRAM APLIKASI EDITOR KATA BAHASA INDONESIA MENGGUNAKAN METODE APPROXIMATE STRING MATCHING DENGAN ALGORITMA LEVENSHTEIN DISTANCE BERBASIS JAVA

Dewi Rokhmah Pyriana, Suprapto,ST.,MT.,Aswin Suharsono,ST.,MT.
Program Studi Teknik Informatika Universitas Brawijaya
dewi.concordes@gmail.com

ABSTRACT

To help the process of checking the spelling mistakes Indonesian word required an application program that can help the work of an author to edit scripts and thesis. In checking the spelling of a word, if found wrong word then it should be possible to search words that correspond with specific string search approach is approximate approach (Approximate String Matching). In the approach used Levenshtein Distance algorithm which has three kinds of operations to transform one string into another string, the removal, insertion, and replacement. These operations are used to calculate distance of the two strings are required for consideration of the suitability of a string with a string source.

The method of analysis used is the structured analysis and design methods using modeling languages DFD (Data Flow Diagram). The testing process consists of two stages, validation testing and accuracy testing. The accuracy testing is used to test the accuracy of the application editor and checking the word sentence structure. Based on the test results of the 10 texts, showed that the system can check the spelling of the word with an alternative justification is accurate and displays the word derived from the word which has the smallest distance value. The results of tests on 10 single sentence shows that the system can also display a single sentence structure Indonesian.

Key words: checking the spelling of word, Levenshtein Distance, the distance values

ABSTRAK

Sampai saat ini, untuk mengetahui kesalahan penulisan kata pada sebuah naskah tulisan atau skripsi masih menggunakan cara manual, yaitu dengan membaca satu per satu kalimat yang ditulis. Cara tersebut akan memakan waktu yang cukup lama dan memiliki akurasi yang rendah, karena tingkat ketelitian yang mungkin terbatas. Untuk membantu mengoreksi kesalahan ejaan kata Bahasa Indonesia digunakan metode *Approximate String Matching* menggunakan algoritma *Levenshtein Distance*. Algoritma *Levenshtein Distance* menghitung jumlah minimum pentransformasian suatu *string* menjadi *string* lain yang meliputi penggantian, penghapusan, dan penyisipan yang akan menghasilkan nilai jarak dari kedua *string*.

Metode analisis yang digunakan adalah metode *structured analysis and design* dengan menggunakan bahasa pemodelan DFD (*Data Flow Diagram*). Implementasi perancangan menggunakan bahasa pemrograman Java. Pengujian aplikasi menggunakan metode *black-box testing* yang meliputi pengujian validasi. Pengujian akurasi digunakan untuk melakukan pengujian terhadap aplikasi editor kata dan pengecekan struktur kalimat. Berdasarkan hasil pengujian terhadap 10 teks, didapatkan hasil bahwa sistem dapat melakukan pengecekan ejaan kata dengan akurat sesuai dengan pencocokan hasil pengecekan secara manual. Hasil pengujian terhadap 10 kalimat tunggal menunjukkan bahwa sistem juga dapat menampilkan struktur kalimat tunggal Bahasa Indonesia.

Kata kunci : kata Bahasa Indonesia, Levenshtein Distance, nilai jarak

I. PENDAHULUAN

1.1 LATAR BELAKANG

Menulis adalah kemampuan seseorang untuk mengungkapkan ide, pikiran, pengetahuan, ilmu dan pengalaman-pengalaman hidupnya dalam bahasa tulis yang runtut, enak dibaca, dan dipahami oleh orang lain. Menulis dapat dilakukan dimana saja dan melalui media apa saja. Naskah tulisan pada media cetak seperti buku, koran, atau majalah akan diedit oleh editor terlebih dahulu sebelum diterbitkan. Penggunaan kata baku yang sesuai dengan kaidah Bahasa Indonesia tidak hanya berlaku di bidang jurnalistik, namun berlaku juga pada penulisan naskah skripsi. Kalimat-kalimat yang digunakan juga harus mengikuti struktur kalimat yang benar sesuai dengan kaidah Bahasa Indonesia.

Kesalahan pengejaan pada editor teks sering kali terjadi. Pemeriksaan kesalahan pengejaan biasa dilakukan ketika tulisan telah selesai dibuat. Pada teks yang pendek, hal tersebut tentu tidak sulit untuk dilakukan. Namun, ketika ukuran teks sudah mencapai lebih dari sepuluh ribu kata atau bahkan jutaan kata, pemeriksaan dengan cara tersebut sangat menyulitkan. [ILM-10]

Perangkat lunak pengolah kata seperti Microsoft Word telah memberikan fasilitas pendukung untuk pemeriksaan ejaan (*spelling and grammar checker*). Akan tetapi, fasilitas tersebut hanya dapat diaplikasikan untuk Bahasa Inggris. Meskipun bisa menambahkan kata sendiri dalam *dictionary list* pada *custom dictionary*, namun hal tersebut tidak menjamin jalannya penggunaan *spelling and grammar checker* untuk kata Bahasa Indonesia.

Untuk membantu proses pengecekan kesalahan ejaan kata Bahasa Indonesia dan mengetahui struktur kalimatnya, diperlukan sebuah program aplikasi yang dapat membantu pekerjaan seorang penulis untuk mengedit naskah tulisan maupun skripsi. Dalam pengecekan ejaan kata, apabila ditemukan kata yang salah maka harus dilakukan pencarian kemungkinan kata yang sesuai. Pada proses pencarian kemungkinan kata tersebut, diperlukan suatu pendekatan pencarian *string* khusus yaitu dengan pendekatan perkiraan (*Approximate String Matching*). [ADI:09]

Approximate String Matching yaitu teknik untuk pencocokkan pola pada *string* dengan cara pendekatan, kinerja dari metode ini tidak harus mirip dengan sebenarnya cukup dengan adanya pendekatan saja. Dalam pendekatan tersebut, ada tiga macam operasi yang digunakan untuk mentransformasikan suatu *string* menjadi *string* yang lain. Operasi tersebut antara lain operasi penghapusan, penyisipan, dan penggantian. Operasi-operasi ini digunakan untuk menghitung jumlah perbedaan yang diperlukan untuk pertimbangan kecocokan suatu *string* dengan *string* sumber. Jumlah perbedaan tersebut diperoleh dari penjumlahan semua pengubahan yang terjadi dari masing-masing operasi. Penggunaan perbedaan tersebut diaplikasikan dalam berbagai macam algoritma, misalnya *Hamming*, *Levenshtein*, *Damerau-Levenshtein*, *Jaro-Winkler*, *Wagner-Fischer*, dan lain-lain. [ADI:09]

Algoritma *Levenshtein*, atau sering disebut dengan *Levenshtein Distance* atau *Edit distance* merupakan algoritma pencarian jumlah perbedaan *string* yang ditemukan oleh Vladimir Levenshtein, seorang ilmuwan Rusia, pada tahun 1965. Algoritma ini digunakan secara luas dalam berbagai bidang, misalnya mesin pencari, pengecek ejaan (*spell checking*), pengenal pembicaraan (*speech recognition*), pengucapan dialek, analisis DNA, pendeteksi pemalsuan, dan lain-lain. [ADI:09]

1.2 RUMUSAN MASALAH

Pada Pembuatan Program Aplikasi Editor Kata Bahasa Indonesia Menggunakan Metode *Approximate String Matching* Dengan Algoritma *Levenshtein*

Distance Berbasis Java ini, masalah-masalah yang akan diselesaikan adalah :

a. Bagaimana mengecek kesalahan ejaan kata pada teks Bahasa Indonesia.
b. Bagaimana menentukan tingkat kemiripan kata menggunakan metode *Approximate String Matching* dengan algoritma *Levenshtein Distance*.
c. Bagaimana mengetahui struktur kalimat tunggal pada teks Bahasa Indonesia.
d. Bagaimana validitas fungsional dan tingkat keakurasian program editor kata Bahasa Indonesia.
e. Bagaimana pengaruh jumlah kata yang diinputkan dan banyaknya jumlah kata yang salah ejaannya terhadap waktu eksekusi.

1.3 BATASAN MASALAH

Dalam pembuatan Program Aplikasi Editor Kata Bahasa Indonesia Menggunakan Metode *Approximate String Matching* dengan Algoritma *Levenshtein Distance* Berbasis Java ini, ada beberapa hal yang dibatasi, yaitu:

a. Program tidak melakukan proses *stemming* (memecah kata menjadi kata dasar).
b. Proses untuk mengetahui struktur kalimat tunggal dengan satu subjek dan satu predikat diambil dari referensi [ART-11].
c. Program hanya menampilkan struktur kalimat tunggal yang terdiri dari S-P, S-P-K, S-P-O, dan S-P-O-K.
d. Program editor kata Bahasa Indonesia dijalankan dengan menggunakan Java *Desktop Application*.
e. Perangkat lunak ini dibuat untuk membantu pengecekan ejaan kata salah hanya pada kata Bahasa Indonesia
f. *Database Management System* yang digunakan adalah MySQL.

1.4 TUJUAN

Pembuatan Program Aplikasi Editor Kata Bahasa Indonesia Menggunakan Metode *Approximate String Matching* Dengan Algoritma *Levenshtein Distance* Berbasis Java ini bertujuan untuk membantu pekerjaan manusia dalam proses pengecekan kesalahan penulisan kata pada sebuah naskah dan mengubahnya menjadi kata yang benar, serta membantu mengetahui struktur kalimat tunggal dalam Bahasa Indonesia.

1.5 MANFAAT

Manfaat dari Program Aplikasi Editor Kata Bahasa Indonesia Menggunakan Metode *Approximate String Matching* Dengan Algoritma *Levenshtein Distance* Berbasis Java ini adalah :

a. Mempercepat proses untuk mengedit sebuah naskah dan menghasilkan hasil tulisan yang sesuai dengan kaidah penulisan Bahasa Indonesia dengan ketelitian yang cukup tinggi.
b. Program yang dibuat tidak hanya dapat digunakan bagi seorang editor yang akan menerbitkan sebuah berita di media cetak maupun elektronik, tetapi dapat juga digunakan untuk membuat naskah karya ilmiah maupun skripsi.

II. DASAR TEORI

2.1 Editor Teks

Editor teks, terutama *word processor* memiliki fasilitas untuk mengecek *spelling* kata-kata yang ada dalam dokumen. Pengecekan ini juga dapat membantu dalam pengoreksian kesalahan pengejaan tersebut. Koreksi hanya dapat dilakukan pada kata yang dikenal bahasanya. Oleh karena itu, diperlukan *database* bahasa yang sesuai.

Algoritma pemrograman dinamis diterapkan pada pengecekan setiap huruf di dalam kata terhadap kata-kata yang terdapat didalam *database*. Namun, pengecekan ejaan kata hanya dilakukan pada kata-kata di dalam kamus yang tidak terlalu jauh perbedaannya sehingga tidak perlu dilakukan perbandingan terhadap

seluruh kata-kata di dalam *database* kamus. [3]

2.2 Klasifikasi Pencocokan *String*

Pencocokan *string* (*string matching*) secara garis besar dapat dibedakan menjadi dua yaitu: [5]

1. *Exact string matching*, merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama.
   Contoh : kata step akan menunjukkan kecocokan hanya dengan kata step.
2. *Inexact string matching* atau *Fuzzy string matching*, merupakan pencocokan *string* secara samar, maksudnya pencocokan *string* dimana *string* yang dicocokkan memiliki kemiripan dimana keduanya memiliki susunan karakter yang berbeda (mungkin jumlah atau urutannya) tetapi *string-string* tersebut memiliki kemiripan baik kemiripan tekstual/penulisan (*approximate string matching*) atau kemiripan ucapan *(phonetic string matching*). *Inexact string matching* masih dapat dibagi lagi menjadi dua yaitu:
   a. Pencocokan *string* berdasarkan kemiripan penulisan (*approximate string matching*) merupakan pencocokan *string* dengan dasar kemiripan dari segi penulisannya (jumlah karakter, susunan karakter dalam dokumen). Tingkat kemiripan ditentukan dengan jauh tidaknya beda penulisan dua buah *string* yang dibandingkan tersebut dan nilai tingkat kemiripan ini ditentukan oleh pemrogram (*programmer*).
      Contoh: compuler dengan compiler, memiliki jumlah karakter yang sama tetapi ada dua karakter yang berbeda. Jika perbedaan dua karakter ini dapat ditoleransi sebagai sebuah kesalahan penulisan maka dua *string* tersebut dikatakan cocok.
   b. Pencocokan *string* berdasarkan kemiripan ucapan (*phonetic string matching*) merupakan pencocokan *string* dengan dasar kemiripan dari segi pengucapannya meskipun ada perbedaan penulisan dua *string* yang dibandingkan tersebut. Contoh step dengan steb dari tulisan berbeda tetapi dalam pengucapannya mirip sehingga dua *string* tersebut dianggap cocok. Contoh yang lain adalah step, dengan steppe, sttep, stepp, stepe.

Exact string matching bermanfaat jika pengguna ingin mencari *string* dalam dokumen yang sama persis dengan *string* masukan. Tetapi jika pengguna menginginkan pencarian *string* yang mendekati dengan *string* masukan atau terjadi kesalahan penulisan *string* masukan maupun dokumen objek pencarian, maka *inexact string matching* yang bermanfaat.

2.3 Algoritma *Levenshtein Distance*

Pada dasarnya, algoritma ini menghitung jumlah minimum pentransformasian suatu *string* menjadi *string* lain yang meliputi penggantian, penghapusan, dan penyisipan. Algoritma ini digunakan untuk mengoptimalkan pencarian tersebut karena sangat tidak efisien jika dilakukan pencarian setiap kombinasi operasi-operasi *string* tersebut. Oleh karena itu, algoritma ini tergolong program dinamis dalam pencarian nilai minimal tersebut.

Untuk menghitung jaraknya (*edit distance*) digunakan matriks (n + 1) × (m + 1) dimana n adalah panjang *string* s1 dan m adalah panjang *string* s2. Berikut dua *string* yang akan digunakan sebagai contoh: [3]

RONALDINHO
ROLANDO

Kedua *string* tersebut memiliki jarak 6, berarti untuk mengubah string RONALDINHO menjadi ROLANDO diperlukan 6 operasi, yaitu:

1. Mensubstitusikan N dengan L
   RONALDINHO → ROLALDINHO
2. Mensubstitusikan L dengan N
   ROLALDINHO → ROLANDINHO
3. Mensubstitusikan I dengan O
   ROLANDINHO → ROLANDONHO
4. Menghapus O
   ROLANDONHO → ROLANDONH
5. Menghapus H
   ROLANDONH → ROLANDON
6. Menghapus N
   ROLANDON → ROLANDO

Dengan menggunakan representasi matriks dapat ditunjukkan pada gambar berikut:

		R	O	N	A	L	D	I	N	H	O
	0	1	2	3	4	5	6	7	8	9	10
R	1	0	1	2	3	4	5	6	7	8	9
O	2	1	0	1	2	3	4	5	6	7	8
L	3	2	1	1	2	3	4	5	6	7	8
A	4	3	2	2	1	2	3	4	5	6	7
N	5	4	3	3	2	2	3	4	5	6	7
D	6	5	4	4	3	3	2	3	4	5	6
O	7	6	5	5	4	4	3	3	4	5	**6**

Gambar 2.1 Matriks yang sudah berisi nilai *edit distance*
Sumber : [3]

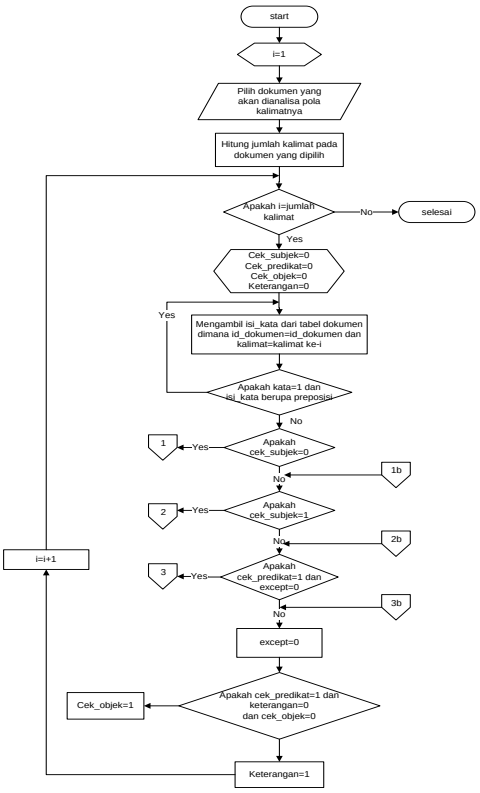
Pada Gambar 2.1, merepresentasikan sebuah tabel matriks yang menunjukkan elemen baris 1 kolom 1 (M[1,1]) adalah jumlah operasi yang diperlukan untuk mengubah *substring* dari kata ROLANDO yang diambil mulai dari karakter awal sebanyak 1 (R) ke *substring* dari kata RONALDINHO yang diambil mulai dari karakter awal sebanyak 1 (R). Sedangkan elemen M[3,5] adalah jumlah operasi antara ROL (*substring* yang diambil mulai dari karakter awal sebanyak 3) dengan RONAL (*substring* yang diambil mulai dari karakter awal sebanyak 5). Elemen terakhir (paling kanan bawah) adalah elemen yang nilainya menyatakan jarak kedua *string* yang dibandingkan

2.4 Pemisahan Rangkaian Kata (*Tokenization*)

Tokenization adalah tugas memisahkan deretan kata di dalam kalimat, paragraf atau halaman menjadi token atau potongan kata tunggal atau *termmed word*. Tahapan ini juga menghilangkan karakter-karakter tertentu seperti tanda baca dan mengubah semua token ke bentuk huruf kecil (*lower case*). [4]

2.5 Proses Menentukan Struktur Kalimat

Tahap-tahap yang dilakukan untuk menentukan struktur kalimat dasar pada Bahasa Indonesia adalah sebagai berikut :



Gambar 2.2 *Flowchart* proses menentukan struktur kalimat
Sumber: [2]

III. METODE PENELITIAN

3.1 Studi Literatur

Dalam pengerjaan artikel ilmiah Pembuatan Program Aplikasi Editor Kata

Bahasa Indonesia Menggunakan Metode *Approximate String Matching* Dengan Algoritma *Levenshtein Distance* Berbasis Java ini, literatur yang harus dicari adalah yang berhubungan dengan editor teks, klasifikasi pencocokan *string*, algoritma *Levenshtein Distance*, pemisahan rangkaian kata (*tokenization*), dan proses menentukan struktur kalimat.

3.2 Identifikasi Masalah

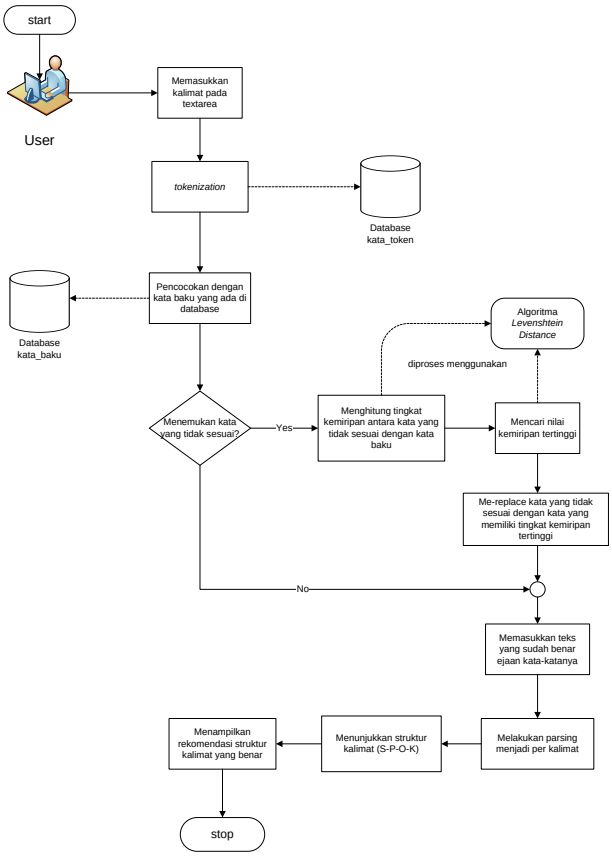
Aplikasi ini dibuat dengan tujuan untuk membantu pekerjaan manusia dalam proses pengecekan kesalahan penulisan kata beserta struktur kalimat pada sebuah naskah dan mengubahnya menjadi kata yang benar. Program pengolah kata yang saat ini ada, masih belum menyediakan menu *spelling checker* Bahasa Indonesia, sehingga untuk proses pengecekan kata yang salah, masih menggunakan cara manual. Program ini nantinya akan bermanfaat bagi seorang editor yang akan menerbitkan media cetak, mahasiswa yang menulis naskah skripsi, dan dosen yang biasanya memeriksa hasil pekerjaan skripsi mahasiswa.

3.3 Analisis Kebutuhan

Analisis kebutuhan bertujuan untuk mendapatkan semua kebutuhan yang diperlukan dari sistem yang akan dibangun. Metode analisis yang digunakan adalah metode *structured analysis and design* dengan menggunakan bahasa pemodelan DFD (*Data Flow Diagram*). *Data Flow Diagram* digunakan untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan satu sama lain dengan alur data, baik secara manual maupun komputerisasi. Analisis kebutuhan dilakukan dengan mengidentifikasi semua kebutuhan (*requirements*) sistem.

3.4 Perancangan

Perancangan aplikasi dilakukan setelah semua kebutuhan sistem didapatkan melalui tahap analisis kebutuhan. Implementasi aplikasi dilakukan dengan mengacu kepada perancangan aplikasi. Implementasi perangkat lunak dilakukan dengan menggunakan bahasa pemrograman berorientasi objek yaitu menggunakan bahasa pemrograman Java (Java *Standard Edition*). Langkah-langkah pembuatan sistem pada artikel ilmiah ini adalah sebagai berikut :



Gambar 3.1 Langkah-langkah pembuatan Program Aplikasi
Sumber : [Metode Penelitian]

3.5 Implementasi

Program aplikasi ini dirancang menggunakan database MySQL berbasis Java menggunakan Netbeans versi 6.8

3.6 Pengujian

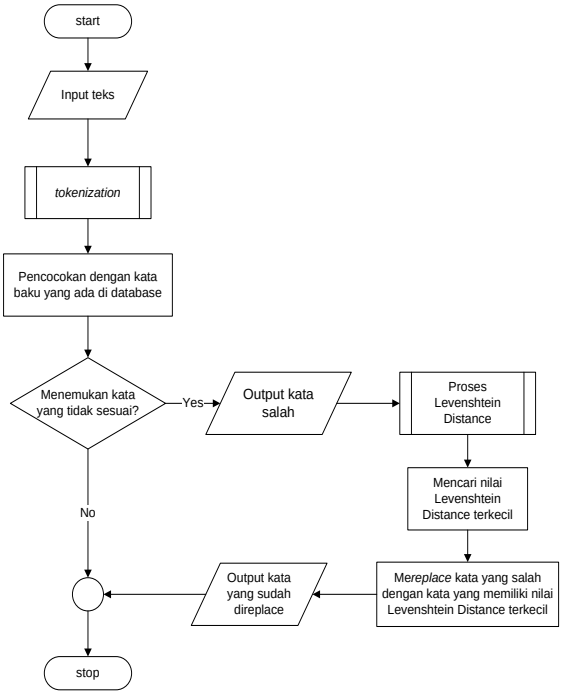
Pada skripsi ini, dilakukan pengujian terhadap cara kerja algoritma *Levenshtein Distance* terhadap proses pencocokan antara kata yang salah ejaannya dengan

masing-masing kata yang sesuai di *database*. Selanjutnya, akan dihitung jarak antara dua kata yang dibandingkan tersebut dan diambil jarak yang terkecil sebagai *ranking* tertinggi. Proses pengujian terdiri dari dua tahap, yaitu pengujijan validasi dan pengujian akurasi. Proses pengujian akurasi dilakukan untuk mengetahui performa dari aplikasi editor kata Bahasa Indonesia. Efisiensi waktu juga akan dihitung karena kata yang salah ejaannya akan dibandingkan dengan ribuan kata yang sudah sesuai di *database*.

IV. PERANCANGAN

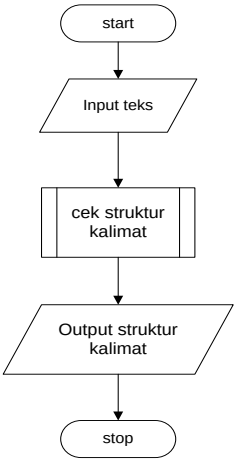
4.1 Perancangan Algoritma

Perangkat lunak pada artikel ilmiah ini memiliki beberapa proses perancangan algoritma yaitu algoritma proses parsing kata (*tokenization*), proses *Levenshtein Distance*, dan algoritma cek struktur kalimat. Secara umum, diagram alir program editor kata adalah sebagai berikut:



Gambar 4.1 Diagram Alir Program Editor Kata
Sumber: [Perancangan]

Secara umum, diagram alir program cek struktur kalimat adalah sebagai berikut:



Gambar 4.2 Diagram Alir Program Cek Struktur Kalimat
Sumber: [Perancangan]

4.2 Rancangan Algoritma Proses *Tokenization*

Proses tokenization dilakukan dengan memecah kalimat menjadi potongan kata dan mengubah semua tanda baca menjadi spasi.

4.3 Rancangan Algoritma Proses *Levenshtein Distance*

Proses *Levenshtein Distance* dilakukan dengan membuat matriks dari dua kata yang dibandingkan (kata yang salah dengan kata baku). Dari setiap kata yang salah dicari jaraknya dengan seluruh kata baku yang ada di database dan didapatkan nilai *Levenshtein Distance*.

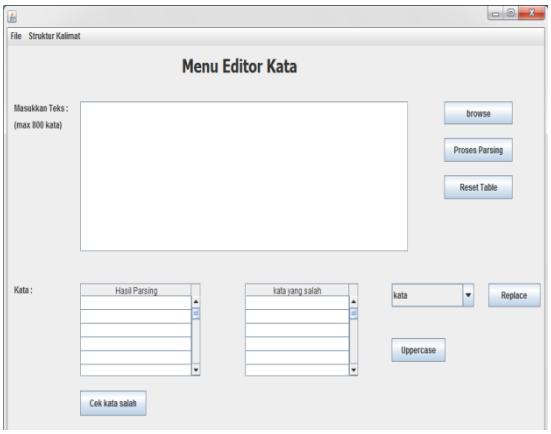
4.4 Rancangan Algoritma Proses Cek Struktur Kalimat

Proses Cek Struktur Kalimat dilakukan dengan menentukan jenis dari tiap-tiap kata dari setiap kalimat. Jenisnya dapat berupa kata benda (n), kata kerja (v), atau kata sifat (adj). Setelah mengetahui jenis katanya, akan ditentukan struktur kalimatnya.

V. IMPLEMENTASI

5.1 Antarmuka Menu Editor Kata

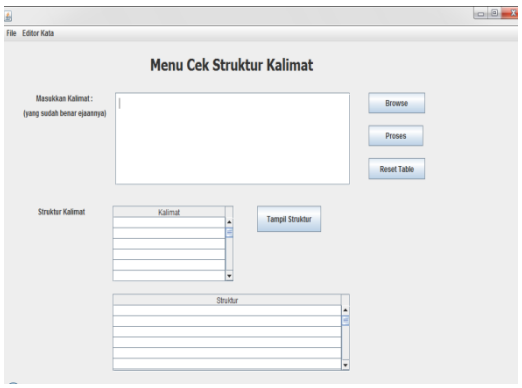
Menu Editor Kata merupakan menu yang dibuat untuk membantu *user* dalam mengecek ejaan kata Bahasa Indonesia. Tampilan Menu Editor Kata dapat dilihat pada Gambar 5.1 :



Gambar 5.1 Tampilan Menu Editor Kata
Sumber: [Implementasi]

5.2 Antarmuka Menu Cek Struktur Kalimat

Menu Cek Struktur Kalimat merupakan menu yang dibuat untuk mengetahui struktur kalimat tunggal Bahasa Indonesia. Tampilan Menu Cek Struktur Kalimat dapat dilihat pada Gambar 5.2 :



Gambar 5.2 Tampilan Menu Cek Struktur Kalimat
Sumber: [Implementasi]

VI. PENGUJIAN DAN ANALISIS

6.1 Pengujian Akurasi

Pengujian akurasi menu editor kata Bahasa Indonesia dilakukan dengan menggunakan 10 teks yang memiliki jumlah kata yang berbeda-beda dan 5 kali pengujian pada teks yang sama dengan jumlah kata salah yang berbeda-beda. Pengujian akurasi untuk menu cek struktur kalimat terdiri dari 10 kalimat tunggal yang memiliki struktur kata yang berbeda-beda. Prosedur pengujiannya adalah memasukkan teks ke dalam sistem aplikasi kemudian dicocokkan dengan pengecekan ejaan kata dan struktur kalimat yang dilakukan secara manual. Hasil akurasi pada menu editor kata dihitung berdasarkan keakuratan hasil alternatif kata yang muncul pada sistem dengan kecocokan hasil deteksi kata yang salah ejaannya. Pada proses pengujian, dihitung juga jumlah kata salah pada teks yang dimasukkan dan waktu eksekusinya. Kata yang terdeteksi salah dapat disebabkan oleh dua hal, pertama kata tersebut merupakan kata asing, singkatan, nama orang, nama kota atau kata yang belum terdaftar di *database* dan yang kedua, kata tersebut memang kata yang salah ejaannya. Perhitungan untuk pengujian akurasi pada menu editor kata dijabarkan pada Tabel 6.1 dan 6.2 berikut :

Tabel 6.1 Hasil Perhitungan Pengujian Akurasi Menu Editor Kata

No	Jumlah kata	Jumlah kata salah (sistem)	Jumlah kata salah (manual)	Alternatif kata pembenaran yang tidak sesuai	Waktu (sistem)	Waktu (manual)	Akurasi
1	130	kata asing/belum terdaftar : 4 salah ejaan : 7	7	-	1 menit	4 menit	100 %
2	242	kata asing/belum terdaftar : 5 salah ejaan : 7	7	-	1,39 menit	6,48 menit	100 %
3	313	kata asing/belum terdaftar : 11 salah ejaan : 7	6	-	2,10 menit	7,11 menit	100 %
4	381	kata asing/belum terdaftar : 16 salah ejaan : **12**	12	-	2,39 menit	8,32 menit	100 %
5	443	kata asing/belum terdaftar : 26 salah ejaan : **13**	10	-	3,50 menit	9 menit	100 %
6	502	kata asing/belum terdaftar : 11 salah ejaan : **17**	13	-	4,03 menit	8,7 menit	100 %
7	532	kata asing/belum terdaftar : 7 salah ejaan : **11**	8	-	4,55 menit	7,49 menit	100 %
8	601	kata asing/belum terdaftar : 29 salah ejaan : **16**	12	-	5,17 menit	9,58 menit	100 %
9	677	kata asing/belum terdaftar : 27 salah ejaan : **19**	15	-	5,24 menit	12,38 menit	100 %
10	729	kata asing/belum terdaftar : 25 salah ejaan : **26**	24	-	5,30 menit	14,48 menit	100 %

Sumber: [Pengujian Dan Analisis]

Tabel 6.2 Hasil perhitungan pengujian akurasi menu editor kata dengan panjang kata yang sama

No	Jumlah kata	Jumlah kata salah (pada sistem)	Alternatif kata pembenaran yang tidak sesuai	Waktu (sistem)	Akurasi
1	276	kata asing/belum terdaftar : 18 salah ejaan : **9**	-	1,56 menit	100 %
2	276	kata asing/belum terdaftar : 7 salah ejaan : **42**	13	1,56 menit	69,04 %
3	276	kata asing/belum terdaftar : 3 salah ejaan : **69**	16	1,56 menit	76,81 %
4	276	kata asing/belum terdaftar : 1 salah ejaan : **106**	30	1,56 menit	71,69 %
5	276	kata asing/belum terdaftar : 3 salah ejaan : **143**	44	1,56 menit	69,23 %
Rata-rata akurasi					**77,35 %**

Sumber: [Pengujian Dan Analisis]

Tabel 6.3 Hasil Perhitungan Pengujian Akurasi Menu Cek Struktur Kalimat

No	Kalimat	Cek Struktur (manual)	Cek Struktur (sistem)	Akurasi
1	Adik menggambar bunga	Subjek Predikat Objek	Subjek Predikat Objek/Keterangan	100 %
2	Ia tidur di kantin	Subjek Predikat Keterangan	Subjek Predikat Objek/Keterangan	100 %
3	Mereka pergi ke Makassar	Subjek Predikat Keterangan	Subjek Predikat Objek/Keterangan	100 %
4	Dia menarik paku dari ban	Subjek Predikat Objek Keterangan	Subjek Predikat Objek Keterangan	100 %
5	Mereka menulis surat kepada panitia	Subjek Predikat Objek Keterangan	Subjek Predikat Objek Keterangan	100 %
6	Dia pedagang	Subjek Predikat	Subjek Predikat	100 %
7	Mereka bekerja dengan semangat	Subjek Predikat Keterangan	Subjek Predikat Objek/Keterangan	100 %
8	Mereka mengatakan hal yang salah	Subjek Predikat Objek Keterangan	Subjek Predikat Objek Keterangan	100 %
9	Planet itu menyerupai bintang	Subjek Predikat Keterangan	Subjek Predikat Objek/Keterangan	100 %
10	Anak itu bermain bola	Subjek Predikat Objek	Subjek Predikat Objek/Keterangan	100 %

Sumber: [Pengujian Dan Analisis]

Berdasarkan hasil pengujian akurasi diatas, dapat disimpulkan bahwa :

1. Sistem editor kata Bahasa Indonesia untuk mengecek kebenaran ejaan kata memiliki akurasi yang cukup tinggi dan memiliki efisiensi waktu yang lebih cepat dibandingkan dengan pengecekan dengan cara manual. Tingkat ketelitian sistem juga lebih tinggi dibandingkan dengan pengecekan secara manual terhadap teks yang memiliki jumlah kata relatif banyak.
2. Waktu yang dibutuhkan untuk eksekusi berbanding lurus dengan jumlah input kata. Semakin banyak kata yang diinputkan, semakin banyak pula waktu yang dibutuhkan untuk eksekusi. Sedangkan jumlah banyaknya kesalahan ejaan kata pada suatu teks tidak berpengaruh pada waktu eksekusi.
3. Agar mendapatkan efisiensi waktu yang tidak terlalu lama (kurang dari 5 menit), maka teks yang dimasukkan maksimal 500-600 kata.
4. Sistem cek struktur kalimat Bahasa Indonesia untuk mengetahui struktur kalimat tunggal Bahasa Indonesia memiliki akurasi 100%.

Pada proses pengujian juga masih ditemukan beberapa kesalahan dan kelemahan antara lain :

1. Beberapa kata yang belum ada dalam *database* kata_baku dikenali sebagai kata salah. Hal ini disebabkan karena belum semua kata berimbuhan masuk ke *database* kata_baku dan sistem tidak melakukan proses *stemming* untuk mendapatkan kata dasarnya. Jadi, apabila daftar kata pada *database* kata_baku dapat dilengkapi lagi, maka akurasi pembenaran kata akan semakin tinggi.
2. Semakin bervariasi kesalahan ejaan kata, maka alternatif kata pembenaran yang muncul mungkin akan tidak sesuai dengan maksud kata yang sebenarnya, karena sistem hanya mencari jarak kata yang terkecil.
3. Pada proses menu editor kata, setelah *user* selesai melakukan *replace* seluruh kata yang salah, user harus mengecek kembali secara manual kata-kata yang pada awalnya tercetak *uppercase*, karena pada proses parsing, seluruh kata diubah menjadi *lowercase*.

Pada proses cek struktur kalimat, untuk beberapa kasus kalimat tidak menghasilkan hasil yang akurat dikarenakan *rule* yang terbatas.

VII. KESIMPULAN

Berdasarkan hasil pengujian dan analisis yang dilakukan terhadap kinerja sistem, dapat diambil kesimpulan sebagai berikut :

1. Pada program ini, proses pengecekan ejaan kata Bahasa Indonesia dilakukan dengan proses tokenization dan pencocokan kata-kata baku yang ada di *database*
2. Tingkat kemiripan kata menggunakan metode *Approximate String Matching* dengan algoritma *Levenshtein Distance* ditentukan dengan cara mencari nilai jarak (*edit distance*) dari kata salah yang terdeteksi dengan semua kata baku yang ada di *database.* Kata yang memiliki nilai jarak yang terkecil akan digunakan sebagai alternatif kata ganti untuk kata yang salah ejaannya.
3. Struktur kalimat tunggal pada Bahasa Indonesia dapat diketahui dengan cara mendeteksi jenis kata yang terdapat dalam satu kalimat beserta letak katanya.
4. Berdasarkan hasil pengujian, program editor kata Bahasa Indonesia untuk mengecek kebenaran ejaan kata memiliki validitas dan akurasi yang tinggi. Pengujian menu cek editor kata memiliki akurasi 77,35% terhadap banyak variasi kata salah. Pada menu cek struktur kalimat dengan 10 kalimat tunggal, akurasi yang dihasilkan 100%.
5. Waktu yang dibutuhkan untuk eksekusi berbanding lurus dengan jumlah input kata. Semakin banyak kata yang diinputkan, semakin banyak pula waktu yang dibutuhkan untuk eksekusi. Sedangkan jumlah banyaknya kesalahan ejaan kata pada suatu teks tidak berpengaruh pada waktu eksekusi.

DAFTAR PUSTAKA

[1] Adiwidya, B.M.D. 2009. *Algoritma Levenshtein Dalam Pendekatan Approximate String Matching*. Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung.

[2] Artrianto, D. 2011. *Aplikasi Penentu Struktur Kalimat Bahasa Indonesia*. Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Gunadarma, Jakarta

[3] Ilmy, M.B.; Rahmi, N.; Bu'ulölö, R.L. 2010. *Penerapan Algoritma Levenshtein Distance untuk Mengoreksi Kesalahan Pengejaan pada Editor Teks*. Laboratorium Ilmu dan Rekayasa Komputasi Departemen Teknik Informatika, Institut Teknologi Bandung, Bandung.

[4] Manning; Christopher D.; Prabhakar R; Schutze; Hinrich. 2008. *Introduction to Information Retrieval*, Cambridge University Press.

[5] Syaroni, M. dan Munir, R. 2005. *Pencocokan String Berdasarkan Kemiripan Ucapan (Phonetic String Matching) Dalam Bahasa Inggris*. Laboratorium Ilmu dan Rekayasa Komputasi Departemen Teknik Informatika, Fakultas Teknologi Industri, Institut Teknologi Bandung, Bandung.