PENGOREKSI KESALAHAN EJAAN BAHASA INDONESIA MENGGUNAKAN METODE LEVENSHTEIN DISTANCE

SKRIPSI

KONSENTRASI REKAYASA KOMPUTER

Diajukan untuk memenuhi persyaratan
memperoleh gelar Sarjana Teknik

Disusun oleh:

RACHMANIA NUR DWITIYASTUTI

NIM. 0710630068-63

KEMENTERIAN PENDIDIKAN NASIONAL

UNIVERSITAS BRAWIJAYA

FAKULTAS TEKNIK

MALANG

2013

PENGANTAR

Puji dan Syukur penulis panjatkan ke hadirat Allah SWT, yang telah melimpahkan segala petunjuk dan rahmat-Nya sehingga penulis dapat menyelesaikan kewajiban menyusun tugas akhir ini. Skripsi berjudul "Pengoreksi Kesalahan Ejaan Bahasa Indonesia Menggunakan Metode Levenshtein Distance" ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik di Jurusan Teknik Elektro Universitas Brawijaya.

Penulis menyadari bahwa dalam penyusunan laporan tugas akhir ini masih terdapat banyak kekurangan, baik dalam materi maupun teknisnya. Hal ini disebabkan pengetahuan dan kemampuan penulis yang terbatas. Karenanya, dengan rendah hati penulis menerima kritik dan saran dari pembaca yang sifatnya membangun dan memperbaiki sehingga dapat mengarah kepada penyusunan yang lebih baik.

Penulis menyadari bahwa penyusunan skripsi ini tidak terlepas dari bantuan berbagai pihak. Oleh karena itu, penulis menyampaikan terima kasih kepada:

1. Bapak dan Ibu yang selalu mendukung dan mendoakan, tidak pernah berhenti mendorong dan member pengertian dalam hingga terselesaikannya skripsi ini,
2. Mbak Dea dan dik Iqbal, serta seluruh keluarga besar atas dukungan dan doa selama ini untuk penulis,
3. Bapak Adharul Muttaqin, ST., MT. dan Bapak Ir. Muhammad Aswin, MT. selaku dosen pembimbing yang telah memberikan bimbingan dan pengarahan-pengarahan dalam proses penyusunan tugas akhir ini, sehingga dapat menambah ilmu bagi penulis.
4. Dr. Ir. Sholeh Hadi Pramono., MS selaku Ketua Jurusan Teknik Elektro Universitas Brawijaya,
5. Seluruh staff pengajar dan karyawan Jurusan Teknik Elektro Universitas Brawijaya Malang,
6. Teman-teman seperjuangan, Ita Dwi Purnamasari, Titis Hayuning W.P., Suci Imani Putri, Rizal Arif Zulfikar, terima kasih atas dukungan dan bantuannya selama ini,
7. Fauzi Fitrah Shidiq, Annisa Kamil, M. Lutfi Abrori, Sintong Arfiansyah, Adityo Nugroho, Wididarma A., Akbar Riyan N., dan Mukhlis Fajar, terima kasih sudah selalu menemani dan memberi dorongan untuk terus maju,

8. Teman-teman lab Komputasi dan Jaringan, terima kasih untuk semuanya..,
9. Teman-teman Core ’07 dan teman-teman paket E ’07 yang selalu memberi motivasi dan dukungan.
10. Semua pihak lain yang tidak mungkin untuk dicantumkan namanya satu-persatu, terima kasih banyak atas segala bentuk bantuan dan dukungannya

Pada akhirnya, penulis menyadari bahwa skripsi ini masih belum sempurna. Oleh karena itu, penulis sangat mengharapkan kritik dan saran yang membangun. Penulis berharap semoga skripsi ini dapat bermanfaat bagi pembacanya.

Malang, Juli 2013

Penulis

ABSTRAK

Rachmania Nur Dwitiyastuti, 2013.PENGOREKSI KESALAHAN EJAAN BAHASA INDONESIA MENGGUNAKAN METODE LEVENSHTEIN DISTANCE. Skripsi,Jurusan Teknik Elektro, Fakultas Teknik, Universitas Brawijaya.
Dosen Pembimbing : Adharul Muttaqin, ST., MT dan Ir. Muhammad Aswin, MT .

Penggunaan komputer, khususnya *word processor*, sebagai alat bantu dalam penulisan bisa meringankan kerja manusia. Meskipun demikian, kadang masih terjadi kesalahan dalam penggunaan *word processor*. Kesalahan yang dimaksudkan adalah kesalahan ejaan yang disebabkan oleh kelalaian pengguna. Kesalahan ejaan terjadi jika kata yang dituliskan pengguna tidak tercantum dalam kamus kata Bahasa Indonesia. Penyebab kesalahan ejaan yang umum terjadi antara lain: penggantian satu huruf, penyisipan satu huruf, penghilangan satu huruf, maupun penukaran dua huruf berdekatan.

Cara untuk mengatasi terjadinya kesalahan ejaan adalah dengan membuat sistem pengoreksi ejaan. Sistem dimulai dengan memeriksa keberadaan kata dalam kamus, lalu memberikan sugesti untuk kata yang tidak terdapat pada kamus. Metode yang digunakan dalam pemberian sugesti adalah metode *Levenshtein Distance*, yaitu metode untuk mencari jumlah perbedaan yang terdapat dalam dua buah *string*.

Pengujian dilakukan dengan memasukkan kata-kata secara acak untuk dikoreksi ejaannya. Dari hasil pengujian, sistem bisa menentukan apakah kata-kata tersebut ejaannya benar atau tidak. Dari hasil pengujian pencarian sugesti didapatkan bahwa sistem bisa mendeteksi kesalahan dan memberi sugesti untuk tiap kategori kesalahan ejaan.

Kata kunci: Kesalahan Ejaan, Pengoreksi Ejaan, Levenshtein Distance, Sugesti.

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Seiring dengan perkembangan teknologi, maka kegunaan komputer dirasa makin besar. Komputer berperan penting dalam mempermudah pekerjaan sehari-hari. Salah satu manfaat utama komputer adalah sebagai alat bantu untuk membuat karya tulis. Berbagai aplikasi seperti Ms. Word, Notepad, maupun Open Office Word sudah ditanamkan pada komputer untuk mempermudah pengguna.

Meskipun berbagai aplikasi sudah tersedia, bukan berarti penggunaannya tanpa masalah. Masalah yang dimaksud adalah adanya kesalahan penulisan ejaan dalam prakteknya, yang seringkali disebabkan karena keteledoran pengguna sendiri. Kesalahan-kesalahan yang terjadi bisa disebabkan oleh ketidaktahuan penulisan, kesalahan yang berhubungan erat dengan posisi tombol papan ketik dan pergerakan jari, dan kesalahan transmisi dan penyimpanan yang berhubungan dengan pengkodean pada jalur mekanisme transmisi data. Kesalahan-kesalahan yang umumnya terjadi antara lain: penggantian satu huruf, penyisipan satu huruf, penghilangan satu huruf, maupun penukaran dua huruf berdekatan.

Untuk mengatasi masalah tersebut, diperlukan suatu metode yang dapat digunakan untuk mempermudah pengguna dalam memeriksa adanya kesalahan ejaan dalam karya tulisnya, sekaligus memberikan alternatif untuk memperbaiki kesalahan tersebut. Pengoreksian ejaan ini dapat dilakukan dengan memberikan ejaan kata yang benar, yaitu dengan memberikan usulan ejaan kata yang mirip dengan kata-kata yang ada di dalam kamus. Metode yang digunakan adalah metode Levenshtein Distance, yaitu suatu pengukuran (metrik) yang dihasilkan melalui perhitungan jumlah perbedaan yang terdapat pada dua string. Untuk mentransformasikan suatu string dengan string lain digunakan tiga pendekatan, yaitu operasi penambahan (*insert*), penghapusan (*delete*), atau penggantian karakter (*substitute*) pada suatu karakter.

Berdasarkan latar belakang masalah tersebut penulis mengangkat judul "Pengoreksi Kesalahan Ejaan Bahasa Indonesia Menggunakan Metode Levenshtein Distance".

1.2 Rumusan Masalah

Berdasarkan latar belakang, maka dapat ditetapkan rumusan masalah sebagai berikut:

1. Bagaimana merancang pendeteksi kesalahan ejaan dengan metode *Levenshtein Distance*
2. Bagaimana mengimplementasi dan menguji pendeteksi kesalahan ejaan dengan metode *Levenshtein Distance*

.

1.3 Batasan Masalah

Mengacu pada rumusan masalah, maka ruang lingkup dari penulisan skripsi ini meliputi:

1. Parameter yang digunakan untuk mendeteksi kata yang benar adalah kata yang terdapat dalam kamus kata Bahasa Indonesia yang tercakup dalam database.
2. Metode yang digunakan untuk mengetahui kebenaran ejaan kata dan mencari sugesti dalam aplikasi ini adalah metode Levenshtein Distance.
3. Database dimuat langsung ke memori.
4. Diasumsikan letak kesalahan kata tidak terdapat pada huruf pertama.

1.4 Tujuan

Tujuan penulisan ini adalah untuk merancang dan membuat pengoreksi kesalahan ejaan Bahasa Indonesia dengan metode *Levenshtein Distance*.

1.5 Sistematika Penulisan

Sistematika penulisan laporan skripsi ini adalah sebagai berikut :

BAB I Pendahuluan

Menjelaskan latar belakang, rumusan masalah, batasan masalah, tujuan, dan sistematika penulisan skripsi

BAB II Tinjauan Pustaka

Menjelaskan kajian pustaka dan dasar teori yang digunakan dalam penyelesaian pembuatan pengoreksi kesalahan ejaan

BAB III Metodologi

Bab ini berisi tentang metode penelitian yang digunakan dalam implementasi sistem pengoreksi kesalahan ejaan.

BAB IV Perancangan dan Implementasi

Menjelaskan langkah langkah perancangan dan hasil implementasi sistem pengoreksi kesalahan ejaan berikut penjelasan algoritma yang digunakan dan penjelasan dari setiap langkah – langkahnya.

BAB V Pengujian dan Analisis

Membahas pengujian dari sistem yang telah dibuat berikut analisa dari hasilnya sehingga dapat diketahui kelebihan dan kekurangan sistem.

BAB VI Kesimpulan dan Saran

Berisi Kesimpulan hasil penulisan dan Saran untuk pengembangan sistem selanjutnya.

BAB II
Tinjauan Pustaka

Bab ini menjelaskan tentang kajian pustaka dan dasar teori yang digunakan untuk menunjang penulisan skripsi dan melandasi pelaksanan penelitian sistem pengoreksi kesalahan ejaan bahasa Indonesia. Beberapa dasar teori yang diperlukan berdasar kajian pustaka untuk penyusunan skripsi ini meliputi kesalahan ejaan, metode Levenshtein Distance, Bahasa Pemrograman C#, dan *Prefix Tree*. Berikut ini adalah paparan sub topik tersebut.

2.1 Kesalahan Ejaan

Ketika seorang pengguna *word processor* mengetikkan sesuatu, kadangkala terdapat kesalahan ejaan di dalamnya. Kesalahan ejaan yang dimaksud adalah jika kata yang dituliskan oleh pengguna tidak ada atau tidak terdaftar pada kamus kata Bahasa Indonesia.

Kesalahan ejaan dapat disebabkan karena beberapa hal, diantaranya:

1. Ketidaktahuan penulisan. Kesalahan ini biasanya konsisten dan kemungkinan berhubungan dengan bunyi kata dan penulisan yang seharusnya.
2. Kesalahan dalam pengetikan yang lebih tidak konsisten tapi mungkin berhubungan erat dengan posisi tombol papan ketik dan pergerakan jari.
3. Kesalahan transmisi dan penyimpanan yang berhubungan dengan pengkodean pada jalur mekanisme transmisi data.

Kesalahan ejaan yang umumnya terjadi antara lain:

1. Penggantian satu huruf
2. Penyisipan satu huruf
3. Penghilangan satu huruf
4. Penukaran dua huruf berdekatan.

Kesalahan ejaan dapat dikoreksi menggunakan dua strategi dasar yang berbeda, yaitu mutlak dan relatif. Secara mutlak, pengoreksian dilakukan dengan membuat suatu tabel variasi ejaan yang salah dengan ejaan yang benarnya. Namun demikian, secara relatif ejaan yang benar dipilih dari kamus yaitu dengan mencari kata dalam kamus yang paling mirip dengan kata yang salah ejaannya.

2.2 Metode Levenshtein Distance

2.2.1 Pengertian Levenshtein Distance

Levenshtein Distance, atau sering disebut juga sebagai edit distance, adalah suatu pengukuran (metrik) yang dihasilkan melalui perhitungan jumlah perbedaan yang terdapat pada dua *string*. Levenshtein distance antara dua *string* didefinisikan sebagai jumlah minimum perubahan yang diperlukan untuk mengganti suatu *string* dengan *string* lain, dengan operasi penambahan (*insert*), penghapusan (*delete*), atau penggantian karakter (*substitute*) pada suatu karakter.

2.2.1.1 *Insertion*

Insertion adalah operasi melakukan penyisipan sebuah karakter ke dalam sebuah *string* tertentu. Misalnya, jika ingin menyisipkan sebuah karakter "a" ke dalam sebuah *string* "kenpa" tepat setelah karakter "n". Dengan dilakukannya *insertion*, *string* yang tadinya "kenpa" akan menjadi "kenapa".

2.2.1.2. *Deletion*

Deletion adalah operasi melakukan penghilangan atau penghapusan sebuah karakter tertentu dari sebuah *string*. Misalnya, jika karakter "n" pada *string* "kenpa" dihapus. Setelah dilakukan *deletion*, *string* akan menjadi "kepa".

2.2.1.3. *Substitution*

Substitution adalah operasi menukarkan sebuah karakter pada *string* tertentu dengan karakter lain. Misalnya, karakter "n" pada *string* "kenpa" ditukar dengan karakter baru "m". Setelah dilakuannya substitution, *string* akan menjadi "kempa".

Yang dimaksud dengan *distance* adalah jumlah perubahan yang diperlukan untuk mengubah suatu bentuk *string* ke bentuk *string* yang lain. Contohnya, *string* "hasil" dan "hasal" memiliki *distance* 1 karena diperlukan satu operasi untuk mengubah *string* "hasal" menjadi "hasil".

2.2.2 Contoh Penggunaan Metode Levenshtein Distance

Contoh lainnya dalam penggunaan metode Levenshtein Distance adalah untuk mengubah *string* "buku" dan "dulu", dibutuhkan dua operasi, yaitu:

1. Mensubstitusikan B dengan D

   BUKU → **D**UKU

2. Mensubstitusikan K dengan L

   DU**K**U → DU**L**U

Dengan menggunakan representasi matrik, langkah-langkah pencarian *distance* antara "buku" dan "dulu" dapat ditunjukkan pada tabel 2.1 sampai tabel 2.5.

		B	U	K	U
	0	1	2	3	4
D	1				
U	2				
L	3				
U	4				

Tabel 2.1 Inisialisasi Matriks

		B	U	K	U
	0	1	2	3	4
D	1	1	2	3	4
U	2				
L	3				
U	4				

Tabel 2.2 Pengisian Nilai *Levenshtein Distance* untuk Baris Pertama

		B	U	K	U
	0	1	2	3	4
D	1	1	2	3	4
U	2	2	1	2	3
L	3				
U	4				

Tabel 2.3 Pengisian Nilai *Levenshtein Distance* untuk Baris Kedua

		B	U	K	U
	0	1	2	3	4
D	1	1	2	3	4
U	2	2	1	2	3
L	3	3	2	2	3
U	4				

Tabel 2.4 Pengisian Nilai *Levenshtein Distance* untuk Baris Ketiga

		B	U	K	U
	0	1	2	3	4
D	1	1	2	3	4
U	2	2	1	2	3
L	3	3	2	2	3
U	4	4	3	3	**2**

Tabel 2.5 Pengisian Nilai *Levenshtein Distance* untuk Baris Keempat

Setelah seluruh matriks terisi nilai, maka *Levenshtein distance* untuk perbandingan *string* "buku" dan "dulu" terletak pada baris ke 4 dan kolom ke 4, yaitu 2.

Jika solusi optimal tersebut ditelusuri maka perubahan yang terjadi adalah dapat dilihat pada tabel 2.6.

		B	U	K	U
	0	1	2	3	4
D	1	1	2	3	4
U	2	2	1	2	3
L	3	3	2	2	3
U	4	4	3	3	**2**

Tabel 2.6 Solusi yang Didapat dari Metode *Levenshtein Distance*

Elemen terakhir (kanan bawah) adalah elemen yang nilainya menyatakan jarak kedua *string* yang dibandingkan. Sehingga algoritma untuk mengisi matriks sesuai dengan yang di atas dapat dilihat pada gambar 2.1.

```
n ← length(s)
m ← length(t)
if n = 0 then return m
else if m = 0 then return n
else
for i = 0 to n do
cost[0][i] ← i
for i = 0 to m do
cost[i][0] ← i
for i = 1 to n do
for j = 1 to m do
if (s[i-1]=t[j-1]) then
cost[j][i] ← 0
else cost[j][i] ← 1
a1 ← cost[j][i-1]+1
a2 ← cost[j-1][i]+1
a3 ← cost[j-1][i-1]+cost[j][i]
cost[j][i] ← min(a1,a2,a3)
return cost[m][n]
```

Gambar 2.1 *Pseudocode* Algoritma *Levenshtein Distance*

Algoritma *Levenshtein Distance* untuk perbandingan kata dalam jumlah yang sangat banyak dapat dioptimasi dengan prinsip branch and bound dengan batas-batas nilai edit distance sebagai berikut :

1. Paling kecil sebesar perbedaan panjang kedua *string* yang dibandingkan
2. Paling besar sebesar panjang *string* yang lebih panjang
3. Nilainya 0 jika dan hanya jika kedua *string* yang dibandingkan identik
4. Jika kedua *string* memiliki panjang yang sama, maka jarak Hamming adalah batas atas dari nilai edit distance

Hasil *Levenshtein distance* yang diperoleh sebenarnya tidak dapat langsung dimanfaatkan, namun perlu diolah untuk memenuhi kebutuhan aplikasi tersebut. Banyak aplikasi yang menggunakan algoritma ini, seperti pengecek ejaan, pemandu penerjemahan, perkiraan dari pengucapan dialek, mesin pencari, pemberi revisi file

dengan membandingkan perbedaan dua buah file, pendeteksi pemalsuan, pengenal percakapan (speech recognition), dan sebagainya.

2.3 C Sharp (C#)

C# merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. C# adalah Java versi Microsoft, sebuah bahasa multi platform yang didesain untuk bisa berjalan di berbagai mesin. C# adalah pemrograman berorientasi Object (OOP). C# memiliki kekuatan bahasa C++ dan portabilitas seperti Java. Fitur-fitur yang diambilnya dari bahasa C++ dan Java adalah desain berorientasi objek, seperti garbage collection, reflection, akar kelas (root class), dan juga penyederhanaan terhadap pewarisan jamak (multiple inheritance).

Bahasa pemrograman C# dibuat sebagai bahasa pemrograman yang bersifat *general-purpose* (untuk tujuan jamak), berorientasi objek, modern, dan sederhana. C# ditujukan agar cocok digunakan untuk menulis program aplikasi baik dalam sistem klien-server (*hosted system*) maupun sistem embedded (*embedded system*), mulai dari program aplikasi yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada program aplikasi yang sangat kecil.

Meskipun aplikasi C# ditujukan agar bersifat 'ekonomis' dalam hal kebutuhan pemrosesan dan memori komputer, bahasa C# tidak ditujukan untuk bersaing secara langsung dengan kinerja dan ukuran program aplikasi yang dibuat dengan menggunakan bahasa pemrograman C.

Fitur-fitur utama dari C# antara lain:

2.3.1 Kesederhanaan C#

Pointer telah dihilangkan dari C#. Hal ini berarti tidak perlu lagi manajemen memori secara manual yang dapat mempersulit programmer dalam membangun suatu aplikasi. Karena berada dalam lingkungan .NET, maka C# memiliki turunan manajemen memori otomatis dan *Garbage Collection*. Untuk menggati pointer, sistem manajemen memori otomatis seperti ini digunakan, hal ini berdampak pada manjemen memori yang lebih mudah dan mempermudah pekerjaan programmer.

C# memiliki dukungan tipe data primitif yang lebih banyak seperti Integer, Floats, dan sebagainya. Dengan adanya tipe data primitif yang lengkap, programmer dapat memilih tipe data yang ingin digunakan

sesuai dengan kondisi yang ada secara optimal tanpa dibatasi oleh jumlah tipe variabel yang disupport.

Pada C#, nilai integer 0 dan 1 tidak diterima lagi sebagai alternatif nilai Boolean. Nilai Boolean adalah murni true atau false. Dengan adanya peraturan ini berarti C# .NET telah menjalankan konsistensinya dalam penggunaan sintaks pemprograman. Hal ini dapat mengurangi ambiguitas yang sering terjadi ketika membangun suatu aplikasi.

2.3.2 Modern

C# didasarkan pada trend yang berkembang saat ini dan sangat ampuh serta mudah digunakan untuk membangun aplikasi yang bersifat interopable, skalabilitas, dan cepat.

C# juga menyertakan dukungan yang dapat membuat setiap komponennya diintegrasikan kedalam web service yang dapat dipanggil dari aplikasi apa saja dan platform apa saja melalui internet.

2.3.3 Berorientasi Objek

C# mendukung enkapsulasi, inheritance, polymorphism, dan interfaces secara penuh. Int, float, dan double bukanlah merupakan object dalam bahasa Java, tetapi C# memperkenalkan struktur yang memungkinkan tipe-tipe primitif dapat menjadi object

2.3.4 *Interoperability*

C# menyertakan dukungan pada COM dan aplikasi berbasis windows. User tidak lagi secara eksplisit mengimplementasikan inteface COM yang tidak diketahui, fitur-fitur seperti itu sudah ada pada C#. C# mengizinkan user untuk menggunakan pointer sebagai blok kode yang bersifat tidak aman. Komponen yang dibuat oleh VB.NET atau bahasa pemrograman lain yang mendukung .NET dapat langsung digunakan oleh C#.

2.3.5 Modular

Kode C# ditulis dengan pembagian masing-masing *class* yang terdiri dari beberapa rutin yang disebut sebagai *member methods*. *Class-Class*

dan metode-metode ini dapat digunakan kembali oleh program atau aplikasi lain. Hanya dengan memberikan informasi yang dibutuhkan oleh Class dan metode yang dimaksud, maka kita akan dapat membuat suata kode yang dapat digunakan oleh satu atau beberapa aplikasi dan program (reusable code).

C# diciptakan dengan harapan menjadi bahasa pemrograman yang simpel, modern, dan juga berbasis OOP. C# berusaha mencapai keungulan yang diharapkan dengan menjadikan developer inti dari Delphi menjadi team leader dari C#. C# menjadi bahasa pemrograman yang unggul karena C# mengadopsi konsep positif dan keunggulan-keunggulan dari bahasa yang lain yang merupakan pendahulu dari

C# seperti C, C++, JAVA, dan juga VB. Konsep OOP misalnya, C# berbasiskan bahasa C++ dan sangat kental dengan pengaruh konsep OOP milik Java. Sehingga dengan pengadopsian bahasa pemrograman yang memang bagus dalam hal pembuatan aplikasi berbasis OOP maka C# pun juga menjadi bahasa yang sering dipakai oleh programmer. Selain itu sokongan dari Microsoft dan juga tool yang ada, serta tidak lupa IDE yang baik membuat C# dapat bersaing dengan bahasa pemrograman lainya.

Dengan segala keunggulan C# .NET mulai dari manajemen memorinya yang lebih baik, standarisasi oleh ECMA dan ISO, dukungan penuh dari vendor, dan lainlain, C# .NET memiliki kelemahan yang mirip dengan keluarga bahasa pemprograman lainnya yang berada di bawah .NET Framework. C# .NET membutuhkan platform yang akan menjalankan programnya sudah terinstal dengan .NET Framework yang besar filenya semakin lama semakin memberatkan end-user.

C# .NET juga mengalami masalah mengenai penggunaan resource komputernya yang lebih banyak daripada bahasa pemprograman lainnya. Hal ini berdampak pada tingginya requirement yang harus dipenuhi oleh suatu platform agar dapat menajalankan aplikasi C# .NET

2.4 Prefix Tree

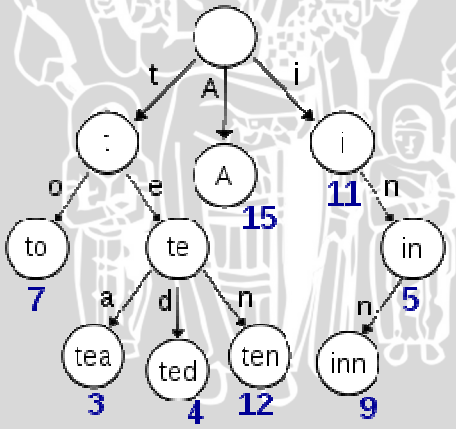
Prefix Tree, atau biasa disebut trie, pertama kali diperkenalkan pada tahun 1960-an oleh Fredkin. Trie berasal dari kata *retrieval* (pengambilan kembali) sesuai dengan fungsinya. Secara etimologi kata ini diucapkan sebagai 'tree'. Meskipun

mirip dengan penggunaan kata 'try' tetapi hal ini bertujuan untuk membedakannya dari general tree.

Dalam ilmu komputer, trie adalah sebuah pohon struktur data yang memerintahkan digunakan untuk menyimpan sebuah array asosiatif di mana kunci biasanya *string*. Tidak seperti Binary Search Tree, tidak ada node di pohon menyimpan kunci yang terkait dengan simpul itu; sebaliknya, posisinya di pohon menunjukkan apa kunci itu terkait dengan.

Pada trie, setiap node pohon merepresentasikan atribut yang telah diuji. Setiap cabang merupakan suatu pembagian hasil uji, dan node daun (leaf) merepresentasikan kategori tertentu. Level node teratas dari sebuah Decision Tree adalah node akar (root) yang biasanya berupa atribut yang paling memiliki pengaruh terbesar pada suatu faktor tertentu.

Pada umumnya trie melakukan strategi pencarian secara *topdown* untuk solusinya. Pada proses mengklasifikasi data yang tidak diketahui, nilai atribut akan diuji dengan cara melacak jalur dari node akar (root) sampai node akhir (daun) dan kemudian akan diprediksi faktor yang dimiliki oleh suatu data baru tertentu. Contoh dari trie ditunjukkan pada gambar 2.2.



Gambar 2.2 *Contoh Prefix Tree*

Simpul teratas dinamakan simpul akar (*root*). Jika suatu simpul memiliki simpul lain di bawahnya, simpul yang lebih atas dinamakan simpul induk (*parent*), sementara simpul yang ada di bawahnya dinamakan simpul anak (*child*). Simpul-simpul paling bawah, yang tidak memiliki anak, sering disebut daun (*leaf*).

BAB III
METODOLOGI PENELITIAN

Bab ini menjelaskan mengenai langkah-langkah yang dilakukan untukmembuat aplikasi pengoreksi kesalahan ejaan bahasa Indonesia. Metode penelitian yang digunakan dalam penyusunan skripsi ini adalah:

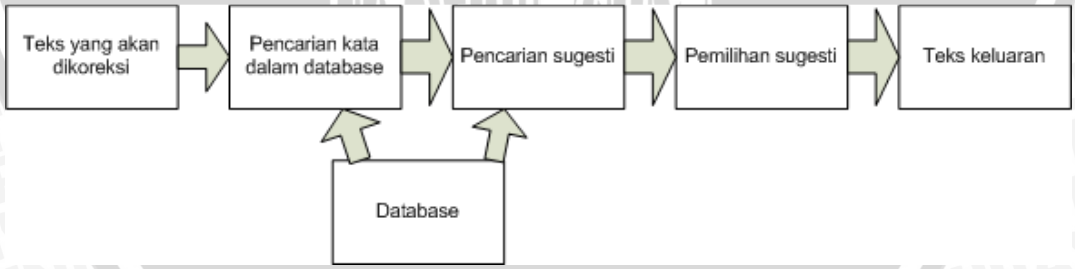
3.1 Studi Literatur

Studi literatur menjelaskan dasar teori yang digunakan untuk menunjang penulisan skripsi. Teori-teori pendukung tersebut meliputi:

1. Mempelajari kesalahan ejaan
2. Mempelajari metode *Levenshtein Distance*
3. Mempelajari *prefix tree*
4. Mempelajari bahasa pemrograman C#

3.2 Diagram Sistem

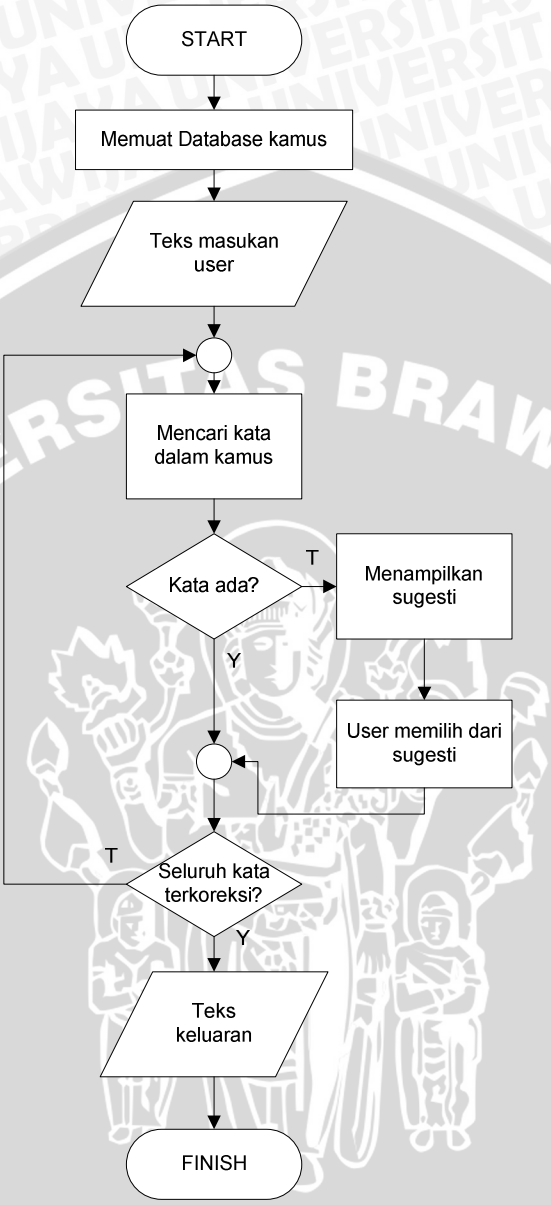
Sistem ini dapat menerima masukan berupa teks yang menggunakan bahasa Indonesia, kemudian teks tersebut di proses didalam sistem yang terkoneksi dengan database sehingga dapat dihasilkan keluaran berupa teks dengan menggunakan bahasa Indonesia dengan ejaan yang benar. Gambar dari diagram sistem tersebut bisa dilihat pada gambar 3.1.



Gambar 3.1 Diagram Sistem

3.3 Cara Kerja Sistem

Diagram alir dari sistem yang akan dibuat dapat dilihat pada gambar 3.2.



Gambar 3.2 Diagram Alir Sistem Pengoreksi Kesalahan Ejaan

Database yang digunakan oleh sistem berupa *file* .txt, terdiri atas 43.092 kata dalam bahasa Indonesia. Untuk mempercepat pencarian data pada database, database tersebut dimuat ke dalam memori, menggunakan *prefix tree* atau *trie*. Database akan otomatis dimuat ke dalam memori saat program pertama kali dijalankan.

Teks masukan user diperoleh dari teks yang terdapat pada *textfield* input. Teks tersebut bisa ditulis langsung pada *textfield* tersebut ataupun dari teks yang sudah dibuat sebelumnya. Setelah teks masukan diambil, kata-kata yang ada dipisahkan dengan melakukan seleksi adanya spasi antar kata. Tiap kata dijadikan sebagai isi *array* dari variabel 'words'. Selanjutnya sistem mencari kata yang sama dalam database untuk setiap indeks *array*.

Apabila kata yang dimasukkan user tidak terdapat dalam kamus, sistem akan menampilkan sugesti. Sugesti didapatkan dari pencarian kata yang memiliki jarak paling dekat dengan kata masukan user. Pencarian dilakukan dengan menggunakan metode *Levenshtein Distance*. Dalam sistem pengoreksi ejaan ini, metode *Levenshtein Distance* digunakan dengan mencari jarak antara kata yang salah dengan kata-kata lain dalam kamus sehingga didapatkan kata-kata yang memiliki jarak dekat (1 atau 2). Kata-kata tersebut selanjutnya digunakan sebagai sugesti yang bisa dipilih user.

Untuk setiap kata yang salah, akan ditampilkan sugesti. User bisa memilih sugesti dari sistem, melakukan pembetulan manual, maupun mengabaikan kesalahan ejaan. Jika user memilih untuk mengabaikan, maka sistem tidak melakukan perubahan pada kata yang salah, dan melanjutkan mengoreksi kata berikutnya. Teks hasil pengoreksian ejaan akan ditampilkan pada *Textfield output*.

3.4 Pengujian dan Analisis

Pengujian dilakukan untuk menjamin dan memastikan bahwa sistem yang dirancang dapat berjalan seperti yang diharapkan. Tolak ukur keberhasilan uji coba adalah semua kebutuhan telah diwujudkan dan berjalan dengan baik. Pengujian yang dilakukan meliputi pengujian pencarian kata, pemberian sugesti, kesesuaian sugesti, dan kompleksitas algoritma.

3.5 Pengambilan Kesimpulan dan Saran

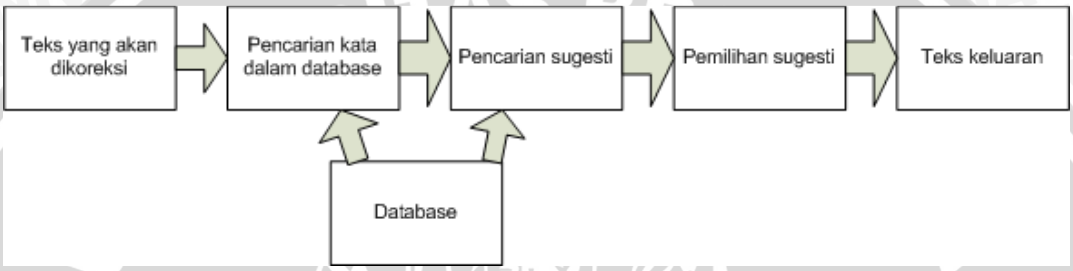
Pengambilan kesimpulan dilakukan setelah semua tahapan perancangan, implementasi dan pengujian sistem aplikasi telah selesai dilakukan. Kesimpulan diambil dari hasil pengujian dan analisis terhadap sistem yang dibangun. Tahap terakhir dari penulisan adalah saran yang dimaksudkan untuk memperbaiki kesalahan-kesalahan yang terjadi dan menyempurnakan penulisan serta untuk memberikan pertimbangan atas pengembangan aplikasi selanjutnya.

BAB IV
PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan perancangan dan implementasi perangkat lunak dari aplikasi pengoreksi ejaan Bahasa Indonesia. Implementasi menggunakan bahasa pemrograman C#.

4.1 Blok Diagram Sistem

Blok diagram sistem menggambarkan garis besar kinerja sistem dalam pengoreksi ejaan bahasa Indonesia.



Gambar 4.1 Diagram Blok Sistem

Berikut adalah penjelasan tiap blok diagram pada Gambar 4.1,

1. Teks yang akan dikoreksi

   Teks ini merupakan teks masukan dari user yang akan dikoreksi ejaannya. User bisa menuliskan teks yang ingin dikoreksi secara langsung pada *textfield* input, ataupun menggunakan teks yang sudah disimpan sebelumnya.

2. Pencarian kata dalam database

   Tiap kata yang telah dimasukkan oleh user akan dibandingkan dengan kata yang ada pada database. Jika kata yang dimasukkan user ada dalam database, maka kata tersebut sudah benar. Bila kata tidak ada dalam database, maka kata tersebut salah dan memerlukan pengoreksian.

3. Pencarian sugesti

   Tiap kata yang salah dibandingkan lagi dengan kata-kata pada database untuk dicari sugesti yang paling mendekati kata masukan.

4. *Database*

   Database dari aplikasi ini berisi daftar kata yang ada di dalam kamus bahasa Indonesia.

5. Pemilihan sugesti

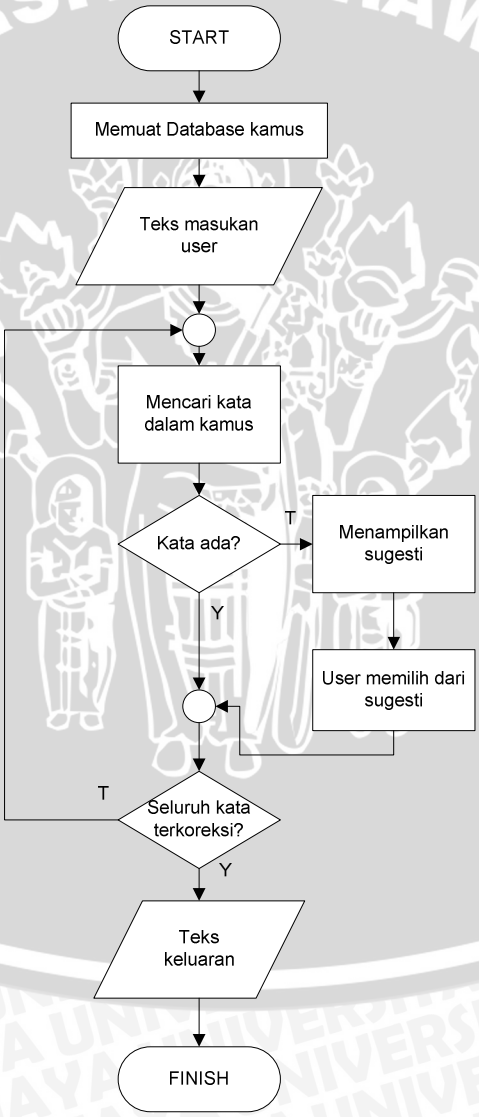
Sistem bisa memberikan lebih dari satu sugesti untuk tiap kata yang salah. User bisa memilih salah satu dari sugesti yang diberikan sistem untuk menjadi teks keluaran, mengabaikan sugesti, maupun melakukan koreksi manual.

6. Teks keluaran

Teks keluaran merupakan teks hasil pengoreksian ejaan.

4.2 Perancangan Sistem Pengoreksi Ejaan

Sistem pengoreksi ejaan berfungsi untuk mengoreksi ejaan dari teks yang dimasukkan oleh user. Diagram sistem pengoreksi ejaan terdapat pada Gambar 4.2



Gambar 4.2 Diagram Sistem Pengoreksi Ejaan

4.2.1 Memuat Database

Database yang digunakan oleh sistem berupa *file* .txt, terdiri atas 43.092 kata dalam bahasa Indonesia. Untuk mempercepat pencarian data pada database, database tersebut dimuat ke dalam memori, menggunakan *prefix tree* atau *trie*. Database akan otomatis dimuat ke dalam memori saat program pertama kali dijalankan. Pada gambar 4.3 ditunjukkan listing program untuk memasukkan kata ke dalam simpul *trie*.

```
private void loadDBButton_Click(object sender, EventArgs e)
    {
        db = true;
        TextReader reader;
        string textLine;
        try
        {
            reader = new StreamReader("DaftarKata.txt");

            while ((textLine = reader.ReadLine()) != null)
            {
                root = Trie.InsertNode(textLine, root);
            }
            reader.Close();
            }
```

Gambar 4.3 *Listing* Program Memuat Database

Untuk membaca *stream* dari database digunakan *StreamReader*. Pembacaannya dilakukan per baris, dengan menggunakan *ReadLine*. Fungsi ini membaca sebaris karakter dari *stream* dan mengembalikannya sebagai data *string*. Pemanggilan *Close()* pada *stream* akan menghilangkan semua data yang ter-*buffer*.

```
public static Node InsertNode(string nama, Node root)
        {
            if (string.IsNullOrEmpty(nama))
                throw new ArgumentNullException("Null Key");
            int index = 1;
            string key;
            Node currentNode = root;
            while (index <= nama.Length)
            {
                key = name[index - 1].ToString();
                Node resultNode = currentNode.Children.GetNodeByKey(key);
                if (resultNode == null)
                {
                    Node newNode = new Node(key, name.Substring(0, index));
                    if (index == nama.Length)
                        newNode.IsTerminal = true;
                    currentNode.Children.Add(newNode);
                    currentNode = newNode;
                }
                else
                {
                    currentNode = resultNode;
                }
                index++;
            }
            return root;
```

Gambar 4.4 Menambahkan Node Pada Trie

Tiap baris yang ada pada file database akan dijadikan sebagai *node* pada Trie. Hal yang pertama kali dilakukan adalah mengecek apakah *string* tersebut bernilai *null* atau tidak dengan menggunakan

method .IsNullOrEmpty. Metode ini menghasilkan kembalian yang bertipe *Boolean* dengan keluaran *true* apabila masukannya *null* atau *string* kosong (“”).

Selanjutnya dilakukan perulangan untuk tiap karakter pada masukan. Jumlah karakter pada masukan dihitung menggunakan *str.length*. Properti *length* berisi integer yang mengindikasikan jumlah karakter pada objek string. Karakter terakhir pada string memiliki indeks -1. Karakter yang didapatkan diubah menjadi bertipe *string*.

Masing-masing *node* harus memiliki nilai yang unik, sehingga dilakukan pengecekan apakah ada node dengan *key* yang sama. Apabila tidak ada, akan dibuat node baru yang berisi karakter yang dimaksudkan.

4.2.2 Teks Masukan User

Teks masukan user diperoleh dari teks yang terdapat pada *textfield* input. Teks tersebut bisa ditulis langsung pada *textfield* tersebut ataupun dari teks yang sudah dibuat sebelumnya. Teks yang digunakan adalah yang memiliki ekstensi rtf. Listing program untuk mengambil teks dari file terdapat pada gambar 4.5.

```
       private void MenuItemOpen_Click(object sender, EventArgs e)
               {
                   string namaFile = "";
              OpenFileDialog fdlg = new OpenFileDialog();
              fdlg.Title = "C# Corner Open File Dialog";
              fdlg.InitialDirectory = @"c:\";
              fdlg.Filter = "RTF File (*.rtf)|*.rtf";
              fdlg.FilterIndex = 2;
              fdlg.RestoreDirectory = true;
              if (fdlg.ShowDialog() == DialogResult.OK)
                    {
                        namaFile = fdlg.FileName;
                    }
                    try
                    {
                        this.rtf1.LoadFile(namaFile);
                    }
                    catch (System.Exception err)
                    {
                        MessageBox.Show("Terjadi   kesalahan   saat
membuka file :\n" + err.Message);
                    }
               }
```

Gambar 4.5 *Listing* program mengambil teks masukan

4.2.3 Pencarian Kata dalam Kamus

Pencarian kata dalam kamus ditujukan untuk menentukan apakah kata yang dimasukkan user benar ejaannya atau tidak. Apabila kata yang dimasukkan oleh user terdapat dalam kamus, maka kata itu sudah benar ejaannya.

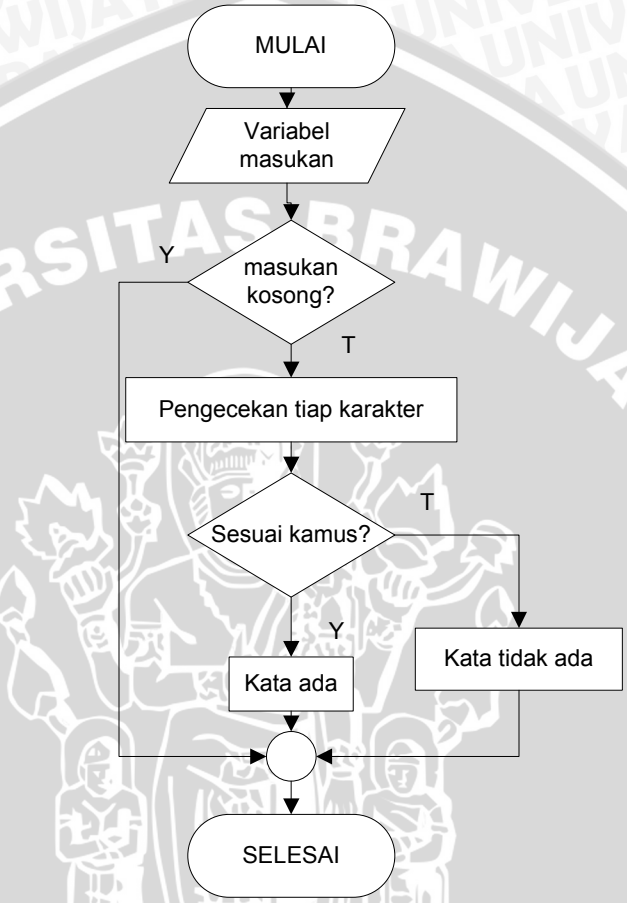
Sebelum melakukan pencarian kata, teks masukan dari user dipisahkan per kata berdasarkan adanya spasi. Kata-kata yang sudah dipisah nantinya dimasukkan ke dalam array.

Pencarian kata dilakukan satu-persatu, dari kata pertama hingga terakhir. Algoritma pencarian kata ditunjukkan pada gambar 4.6

```
public static bool Find(Node node, string key)
        {
           if (string.IsNullOrEmpty(key))
              return true;
           string first = key.Substring(0, 1);
           string tail = key.Substring(1);
           Node curNode = node.Children.GetNodeByKey(first);
           if (curNode != null )
           {
              return Find(curNode, tail);
           }
           else
           {
              return false;
           }
        }
```

Gambar 4.6 *Listing* Program Pencarian Kata

Awalnya kata yang masuk diperiksa dahulu, apakah sudah benar atau belum. Selanjutnya, dilakukan pencocokan antara tiap karakter pada kata dengan tiap node. Diagram alir untuk pencarian kata ditunjukkan pada gambar 4.7.

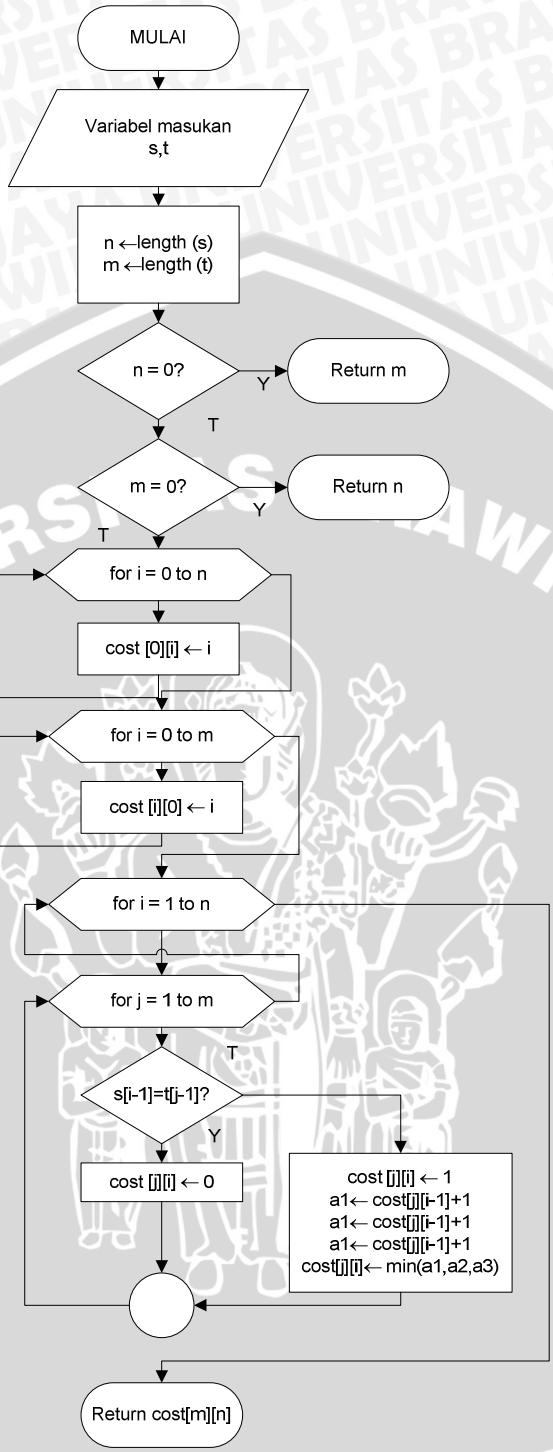


Gambar 4.7 Diagram Alir Pencarian Kata

4.2.4 Penampilan Sugesti

Apabila kata yang dimasukkan user tidak terdapat dalam kamus, sistem akan menampilkan sugesti. Sugesti didapatkan dari pencarian kata yang memiliki jarak paling dekat dengan kata masukan user.

Pencarian dilakukan dengan menggunakan metode *Levenshtein Distance*. Diagram alir dari algoritma Levenshtein Distance ditunjukkan pada gambar 4.8.



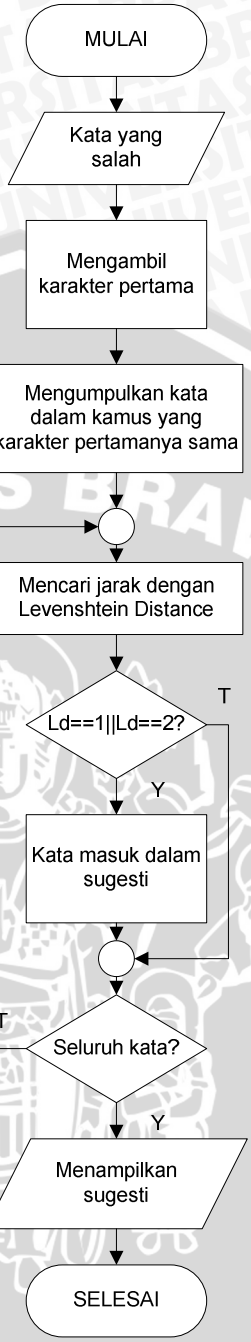
Gambar 4.8 Diagram Alir *Levenshtein Distance*

Keluaran dari algoritma di atas berupa jarak antara dua buah string. *Listing* program *Levenshtein Distance* dapat dilihat pada gambar 4.9.

```
public static int Tes(string a, string b)
        {
            int n = a.Length;
            int m = b.Length;
            int[,] d = new int[n + 1, m + 1];
            int a1, a2, a3;
    if (n == 0)
            {       return m;           }
            if (m == 0)
            {       return n;           }
            for (int i = 0; i <= n; i++)
            {       d[i, 0] = i;            }
            for (int j = 0; j <= m; j++)
            {       d[0, j] = j;            }
            for (int i = 1; i <= n; i++)
            {for (int j = 1; j <= m; j++)
               { int jum = (b[j - 1] == a[i - 1]) ? 0 : 1;
                  a1 = d[i - 1, j] + 1;
                  a2 = d[i, j - 1] + 1;
                  a3 = d[i - 1, j - 1] + jum;
                  d[i, j] = Math.Min(Math.Min(a1, a2), a3);  } }
            return d[n, m];
      }
```

Gambar 4.9 *Listing* Program *Levenshtein Distance*

Dalam sistem pengoreksi ejaan ini, metode *Levenshtein Distance* digunakan dengan mencari jarak antara kata yang salah dengan kata-kata lain dalam kamus sehingga didapatkan kata-kata yang memiliki jarak dekat (1 atau 2). User bisa memilih kata yang dibutuhkan dari sugesti yang diberikan, mengabaikan sugesti, atau mengganti kata yang salah secara manual. Diagram alir pemberian sugesti dapat dilihat pada gambar 4.10.



Gambar 4.10 Diagram Alir Pemberian Sugesti

Terdapat dua cara yang bisa dilakukan untuk menampilkan sugesti. Cara pertama adalah dengan menekan tombol cek pada aplikasi pengoreksi ejaan. Setelah tombol cek ditekan, maka kata-kata yang salah beserta sugestinya akan ditampilkan pada *combo box*. Listing program pemberian sugesti untuk cara pertama ini ditunjukkan pada gambar 4.11.

```
private void sugesti(int arr)
        {   comboBox1.Items.Clear();
            int jmlSala = 0;
            string salah = rtf3.Text;
            string[] kataSalah = salah.Split(' ');
            foreach (string sala in kataSalah)
            {       if (sala != "")
                    jmlSala = jmlSala + 1;            }
            string[] ks = new String[jmlSala];
            int a = 0;
            foreach (string sala in kataSalah)
            {
                if (sala != "")
                {
                    ks[a] = sala;
                    a += 1;
                }
                string teks = "";
                for (int i = 0; i < jmlSala; i++)
                {
                    if (i == 0)
                        teks = ks[i];
                    else
                        teks = teks + " " + ks[i];
                }
                rtf4.Text = arr+ " ";
                if (jmlSala > 0)
                {
                    comboBox1.Text = ks[arr];
                    textBox2.Text = ks[arr];
                    if (string.IsNullOrEmpty(ks[arr]))
                    { }
                    else
                    {
                        int jml = 0;
                        string awal = ks[arr].Substring(0, 1);
                        GetChildrenStartingFromKey(root,      awal,
ks[arr]);
                        string[] sugesti = rtf5.Text.Split(' ');
                        foreach (string kata in sugesti)
                        {
                            jml = jml + 1;
                        }
                        if (rtf5.Text != "")
                        {
                            for (int c = 0; c <jml; c++)
                            {
                                comboBox1.Items.Add(sugesti[c]);
                            }
                            break;
                        }
                    }
                }
                else
                    panel1.Visible = false;
            }
```

Gambar 4.11 *Listing* Program Pemberian Sugesti 1

Cara kedua adalah dengan melakukan klik kanan pada kata yang salah. Sugesti-sugesti yang diberikan pada kata yang salah akan ditampilkan pada *context menu*. *Listing* program pemberian sugestinya dapat dilihat pada gambar 4.12.

```
private void sugesti2()
        {
            contextMenuStrip1.Items.Clear();
            comboBox1.Items.Clear();
            string teksSalah = rtf1.SelectedText;
            bool fnd = Trie.Find(root, teksSalah);
            if (fnd)
            {}
            else
            {
                contextMenuStrip1.Items.Add("Abaikan");
                comboBox1.Text = teksSalah;
                textBox2.Text = teksSalah;
                if (string.IsNullOrEmpty(teksSalah))
                { }
                else
                {
                    int jml = 0;
                    string awal = teksSalah.Substring(0, 1);
                    GetChildrenStartingFromKey(root,         awal,
                     teksSalah);
                    string[] sugesti = rtf5.Text.Split(' ');
                    foreach (string kata in sugesti)
                    {
                        jml = jml + 1;
                    }
                    if (rtf5.Text != "")
                    {
                        for (int c = 0; c < jml; c++)
                        {
                         contextMenuStrip1.Items.Add(sugesti[c]);
                         comboBox1.Items.Add(sugesti[c]);
                        }
                    }
                    contextMenuStrip1.Items.Add("Kata Lain");
                }
            }
        }
```

Gambar 4.12 *Listing* Program Pemberian Sugesti 2

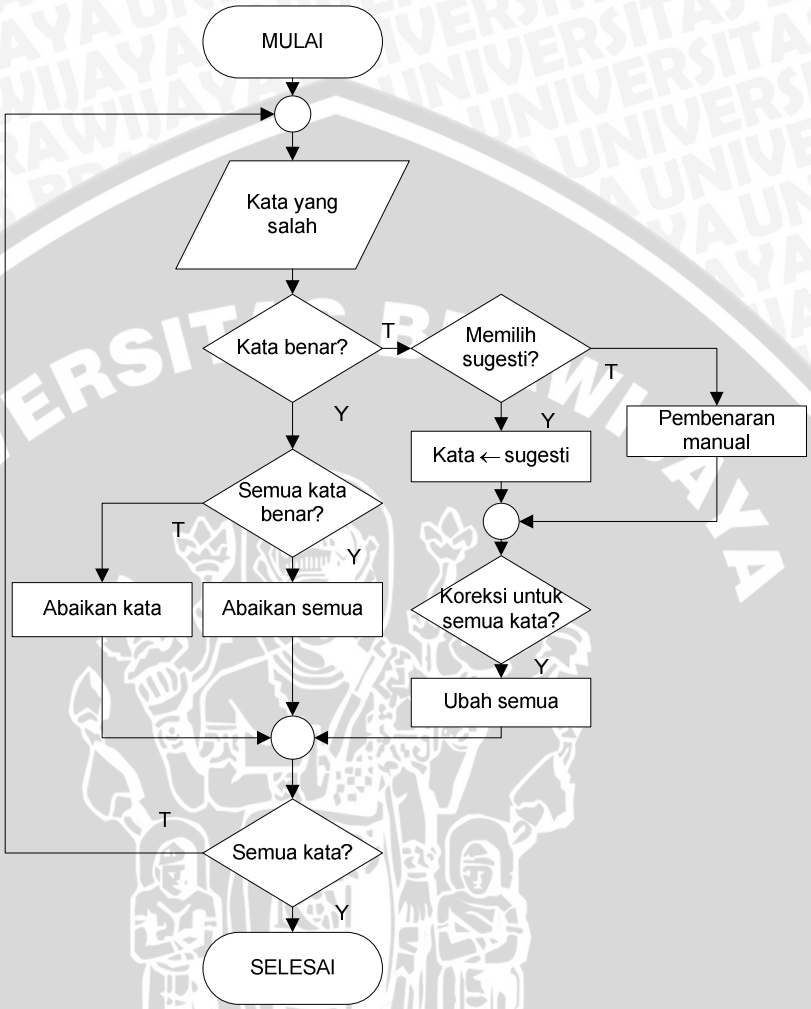
Untuk setiap kata yang salah akan dicari sugesti kata yang paling mendekati. Untuk menyempitkan jangkauan pemilihan sugesti, dianggap bahwa karakter pertama kata sudah benar, sehingga pencarian sugesti dilakukan pada kata-kata yang memiliki huruf awal sama. Proses pencariannya ditunjukkan pada gambar 4.13.

```
public void GetChildrenStartingFromKey(Node node, string key, string kata)
    {
        string[] a = new string[100];
        int n = 0;
        int b = 0;
        Node curNode = Trie.GetChildNode(node, key);

        if (curNode != null)
        {
            IEnumerator iterator = curNode.GetDepthNodes();

            while (iterator.MoveNext())
            {
                Node childNode = (Node)iterator.Current;
                if (childNode.IsTerminal)
                {
                    string s = childNode.Value;
                    int z = Tes(kata, s);
                    if (z == 1 || z==2)
                    {
                        a[n] = s;
                        n = n + 1;
                        if (n == 16)
                            break;
                    }
                    string aa = "";
                    rtf5.Text = a[0];
                    for(b=0;b<n;b++)
                    {
                        if (b == 0)
                            aa = a[0];
                        else
                            aa = aa + " " + a[b];
                    }
                    rtf5.Text = aa;

                }}}}
```

Gambar 4.13 *Listing* Program Pencarian Sugesti

Awalnya program menunjuk node yang memiliki nilai sesuai, yaitu yang memiliki huruf awal sama. Jika *child* dari node itu membentuk kata yang lengkap, maka nilainya akan masuk ke dalam variabel *s*. Nilai variabel s itu nantinya dibandingkan dengan kata yang salah menggunakan metode Levenshtein Distance. Apabila jarak antara dua buah string yang dibandingkan bernilai 1 atau 2, maka *s* akan dimasukkan ke dalam sugesti.

Diagram alir untuk pengoreksian kata oleh user ditunjukkan pada gambar 4.14.



Gambar 4.14 Diagram alir pengoreksian kata

Jika user memilih dari sugesti yang diberikan sistem ataupun memberikan pembetulan manual, maka hasil keluaran akan berubah sesuai pilihan user. *Listing* program untuk pembetulan kata ditunjukkan di gambar 4.15.

```
private void ubahButton_Click(object sender, EventArgs e)
    {
        string ubah="";
        int jml = 0;
        string kata=textBox2.Text;
        if (radioButton1.Checked)
            ubah = comboBox1.SelectedItem.ToString();
        else
            ubah = textBox1.Text;
        string baru=rtf2.Text;
        string[] cari = baru.Split(' ');
        foreach (string a in cari)
        {
            jml = jml + 1;
        }
        for (int j = 0; j < jml; j++)
        {
            if (string.Equals(kata, cari[j], StringComparison.
OrdinalIgnoreCase))
            {
                cari[j] = ubah;
                //MessageBox.Show(cari[j]);
                break;
            }
        }
        string nu = "";
        for (int k = 0; k < jml; k++)
        {
            if (k == 0)
                nu = cari[k];
            else
                nu = nu + " "+cari[k];
        }
        rtf2.Text = nu;

    }
```

Gambar 4.15 *Listing* Program Pemilihan Sugesti

Awalnya sistem mengecek, apakah user memilih sugesti dari sistem atau melakukan pembetulan manual. Apabila yang diambil adalah sugesti sistem, maka kata yang dipilih user akan menjadi isi variabel ubah. Jika tidak, maka kata yang dimasukkan user yang akan menjadi isi variabel ubah.

Sistem mencari kata yang salah, lalu mengubahnya menjadi kata yang diinginkan user. Dalam pencarian kata, sistem menggunakan fungsi string.Equal. Fungsinya adalah untuk membandingkan antara dua buah

string. Apabila bernilai benar, maka kedua string yang dibandingkan itu bernilai sama.

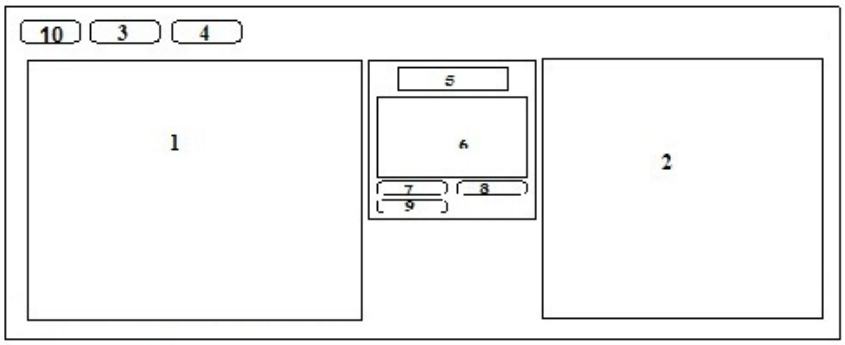
Jika user memilih untuk mengabaikan, maka sistem tidak melakukan perubahan pada kata yang salah, dan melanjutkan mengoreksi kata berikutnya. Sistem menyimpan kata yang diabaikan ke dalam database sementara.

4.3 *User Interface* Sistem Pengoreksi Ejaan

Fungsi yang terdapat dalam Interface Pengoreksi Ejaan meliputi:

1. Judul Program, digunakan agar dapat mengetahui nama program yang berjalan.
2. *Rich TextBox* masukan, berfungsi untuk meletakkan teks masukan yang akan dikoreksi dari user.
3. *Rich TextBox* keluaran, berfungsi untuk meletakkan teks hasil pengoreksian.
4. *Panel* sugesti, berfungsi untuk menampilkan kesalahan-kesalahan ejaan beserta sugestinya masing-masing. Pada panel ini terdapat pilihan ubah, ubah semua, dan abaikan. Tombol ubah dan ubah semua berfungsi untuk mengubah kata yang salah menjadi kata pilihan user. Tombol abaikan berfungsi untuk mengabaikan kata yang salah.
5. Menu, *terdiri* atas menu new, open, save, dan exit. Menu new berfungsi untuk mengembalikan *RichTextBox* masukan dan keluaran ke keadaan kosong. Menu open berfungsi untuk membuka file tes.rtf. Menu save berfungsi untuk menyimpan isi *RichTextBox* keluaran ke dalam tes.rtf.
6. Tombol cek, berfungsi untuk mengecek ejaan dari masukan user.
7. Tombol tandai, berfungsi untuk menandai kata-kata yang memiliki kesalahan ejaan.

Gambar 4.16 adalah gambar bentuk rancangan interface sistem pengoreksi ejaan.

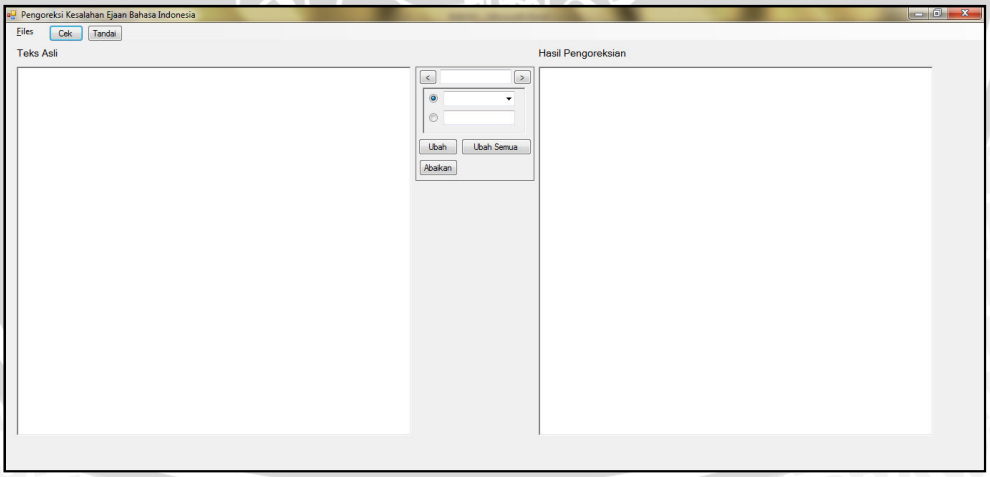


Keterangan gambar:

1. Teks Masukan
2. Teks Keluaran
3. Tombol Tandai
4. Tombol Cek
5. Kata yang salah
6. Sugesti
7. Tombol ubah
8. Tombol ubah semua
9. Tombol abaikan
10. Menu

Gambar 4.16 Rancangan Interface Sistem Pengoreksi Ejaan

Sehingga diperoleh hasil implementasi dari perancangan sistem Pengoreksi Kesalahan Ejaan seperti pada Gambar 4.17



Gambar 4.17 Implementasi Pengoreksi Kesalahan Ejaan

BAB V
PENGUJIAN DAN ANALISIS

Pada bab ini dibahas mengenai pengujian dan analisis sistem dimana akan terlihat kekurangan – kekurangan pada perangkat lunak untuk selanjutnya diadakan pengembangan sistem.

5.1 Pengujian dan Analisis terhadap Pencarian Kata pada Kamus

Pengujian sistem terhadap pencarian kata dalam kamus dilakukan dengan cara memasukkan 14 kata berbeda dengan kemungkinan kata tersebut ejaannya benar atau salah. Parameter keberhasilan pada sistem adalah kesesuaian antara keberadaan kata dalam kamus dengan tampilan kata yang salah pada sistem

Pada Tabel 5.1 ditunjukkan contoh 14 kata untuk diuji kebenarannya.

No	Kata	Kata Salah	Ada di Kamus
1	ABRI	Tidak	Ada
2	cuti	Tidak	Ada
3	Ekuador	Tidak	Ada
4	fyord	Tidak	Ada
5	HAM	Tidak	Ada
6	juz	Tidak	Ada
7	Labuhan	Tidak	Ada
8	nyonya	Tidak	Ada
9	P.T.	Tidak	Ada
10	ruyak	Tidak	Ada
11	Tabanas	Tidak	Ada
12	vulkanologi	Tidak	Ada
13	xenofobia	Tidak	Ada
14	zuriat	Tidak	Ada

Tabel 5.1 Tabel Pengujian Pencarian Kata dalam Kamus

Dari pengujian pencarian kata, sistem sudah dapat memeriksa keberadaan kata pada kamus. Jika kata terdapat dalam kamus, kata tersebut dinyatakan sebagai kata yang benar ejaannya.

5.2 Pengujian dan Analisis Terhadap Pemberian Sugesti

Terdapat enam kemungkinan penyebab terjadinya kesalahan ejaan. Beberapa kesalahan ejaan yang mungkin terjadi pada penulisan beserta sugesti yang diberikan oleh sistem ditunjukkan pada tabel 5.2.

No	Kategori	Kata	Yang Diinginkan	Sugesti
1	Penggantian satu huruf	dakam	dalam	dada, dadah, dadak, dadap, dadar, dahak, daham, dahan, dakar, daki, dakian, daku
2	Penyisipan satu huruf	dallam	dalam	daham, dalal, **dalam**, dalami, daulat
3	Penghilangan satu huruf	dalm	dalam	da, dada, dadu, dagu, daham, dahi, dai, daki, daku, dalai, **dalam**, dalih
4	Penukaran dua huruf	dalma	dalam	dada, dakwa, dalal, **dalam**, dalaman, dalami, dalih, dalil, dam, damai, damar, dana
5	Penggantian dua huruf	ropawsn	rupawan	**rupawan**
6	Kesalahan lebih dari 2 huruf	ruoaawn	rupawan	-

Tabel 5.2 Kategori Kesalahan Ejaan Beserta Contohnya

Pengujian pencarian sugesti dilakukan dengan memasukkan 100 kata ke dalam sistem, dimana di dalamnya terdapat kesalahan-kesalahan yang mewakili tiap kategori penyebab kesalahan ejaan. Hasil pengujian pencarian sugesti ditunjukkan pada Tabel 5.3 dan 5.4.

Jumlah Kata	100
Jumlah Kesalahan	22
Sugesti yang Sesuai	18

Tabel 5.3 Hasil Pengujian Pencarian Sugesti

No	Kata yang Salah	Kategori Kesalahan	Sugesti Sesuai
1	kearamaia	2, 3	ya
2	menberi	1	ya
3	aoa	1	tidak
4	berysaha	1	ya
5	tpi	3	ya
6	mssij	5	ya
7	kadag	3	ya
8	beroikir	1	ya
9	kanapa	1	ya
10	yerjadi	1	tidak
11	padahaal	2	ya
12	slealu	4	ya
13	menciba	1	ya
14	inyik	6	tidak
15	selallu	2	ya
16	piliahn	4	ya
17	bahjan	1	ya
18	menebah	1	tidak
19	waena	1	ya
20	malha	4	ya
21	leih	3	ya
22	katena	1	ya

Tabel 5.4 Pengujian Pencarian Sugesti

Dari hasil pengujian pencarian sugesti didapatkan bahwa sistem sudah bisa mendeteksi kesalahan dan memberi sugesti untuk tiap kategori kesalahan ejaan. Meskipun demikian, tidak semua kesalahan memiliki sugesti yang sesuai. Hal ini disebabkan karena tidak ada kata dalam kamus yang memiliki jarak maksimal 2 dengan kata yang dimasukkan user.

Ketidak seuaian sugesti bisa terjadi karena jumlah kata yang mirip terlalu banyak, sementara yang ditampilkan maksimal 16 sugesti. Biasanya hal ini terjadi jika kata yang dimasukkan terlalu sederhana, sehingga kemungkinan ada kata yang mirip besar. Penyebab lain dari terjadinya eror adalah karena kata yang dimasukkan terlalu kompleks.

5. 3 Pengujian Kompleksitas Algoritma

Kompleksitas waktu dari algoritma Levenshtein Distance berdasarkan analisis program adalah O(mn), dimana m adalah panjang dari *string* pertana dan n adalah panjang dari *string* kedua. Jika kedua *string* memiliki panjang yang sama, maka kompleksitasnya adalah $O(n^2)$.

Waktu yang diperlukan program untuk menghitung jumlah perbedaan antara dua buah *string* dengan panjang *string* pertama dinyatakan sebagai m dan panjang *string* kedua dinyatakan sebagai n terdapat pada tabel 5.5. Pengujian dilakukan sebanyak 15 kali dengan memasukkan 15 kata secara acak untuk dibandingkan dengan 15 kata lainnya dan dicari waktu penyelesaian algoritmanya.

No	String 1	String 2	m	n	Waktu (ms)
1	ah	ha	2	2	0,00097
2	aku	ane	3	3	0,00117
3	palu	aku	4	3	0,00125
4	satu	saru	4	4	0,00141
5	bukan	bulir	5	5	0,00153
6	bukan	buatan	5	6	0,00178
7	bukan	bulatan	5	7	0,00190
8	halaman	ha	7	2	0,00125
9	rahasia	rahang	7	6	0,00198
10	kemarin	kemarin	7	7	0,00182
11	kompleksitas	keluarga	12	8	0,00222

12	peringatan	pengingat	10	9	0,00243
13	musyawarah	mushawarah	10	10	0,00283
14	kompleksitas	komplekstas	12	11	0,00291
15	penanggulangan	penanggulangan	14	14	0,00384
			∑ waktu		0.02929

Tabel 5.5 Waktu Penyelesaian Algoritma

Rata-rata waktu yang diperlukan untuk menghitung perbedaan dari dua buah *string* adalah 0,00195 ms.

BAB VI

PENUTUP

6.1 Kesimpulan

Berdasarkan hasil perancangan, implementasi, pengujian dan analisis yang dilakukan, maka diambil kesimpulan sebagai berikut:

1. Berdasarkan hasil pengujian pada sistem pengoreksi kesalahan ejaan, sistem sudah dapat memeriksa keberadaan kata pada kamus dan memberikan sugesti untuk kata yang salah dengan menggunakan metode *Levenshtein Distance*.
2. Berdasarkan hasil pengujian pencarian kata, sistem bisa mendeteksi 100% kesalahan kata.
3. Dari hasil pengujian pencarian sugesti didapatkan bahwa sistem sudah bisa mendeteksi kesalahan dan memberi sugesti untuk tiap kategori kesalahan ejaan.

6.2 Saran

Saran yang dapat diberikan untuk pengembangan sistem pengoreksi ejaan ini antara lain :

1. Kosa kata dalam kamus dilengkapi sehingga pengoreksiannya bisa lebih tepat.
2. Digunakan jenis-jenis ekstensi file lainnya pada masukan dan keluaran sistem.

DAFTAR PUSTAKA

Hariyanto, Bambang. 2003. *Struktur Data, Memuat Dasar Pengembangan Orientasi Objek*. Bandung: Informatika Bandung.

Nugroho, Adi. 2009. *Algoritma dan Struktur Data dengan C#.* Yogyakarta: CV Andi OFFSET.

Pitts, Robert I. Trie. http://www.cs.bu.edu/teaching/c/tree/trie/. Diakses pada tanggal Diakses pada tanggal 13 Juni 2012 pukul 19.00 WIB.

Rahmat. 2005. Database Management System (DBMS). http://blog.re.or.id/ database-management-system-dbms.htm. Diakses pada tanggal 1 Maret 2011 pukul 17.50 WIB.

Suarga. 2006. *Algoritma Pemrograman*. Yogyakarta: CV Andi OFFSET.

Trevisan, L. 2001. Notes for Lecture 13 – Edit Distance. http://www.cs.berkeley.edu/ ~luca/cs170/notes/lecture13.pdf, Diakses pada tanggal 1 Maret 2011 pukul 17.25 WIB.

Wagito. 2003. *Pemrograman Berorientasi Objek*. Yogyakarta: Gava Media.

Wikipedia. Basis Data. http://id.wikipedia.org /wiki/Basis_Data. Diakses pada tanggal 1 Maret 2011 pukul 17.35 WIB.

Wikipedia. 1999. Levenshtein Distance. http:// en.wikipedia.org/wiki/Edit_distance. Diakses pada tanggal 1 Maret 2011 pukul 17.30 WIB.

Wikipedia. Sistem Manajemen Basis Data. http://id.wikipedia.org/wiki/ Sistem_manajemen_basis_data. Diakses pada tanggal 1 Maret 2011 pukul 17.45 WIB.

Wikipedia. Trie. http:// en.wikipedia.org/wiki/Trie. Diakses pada tanggal 13 Juni 2012 pukul 19.30 WIB.

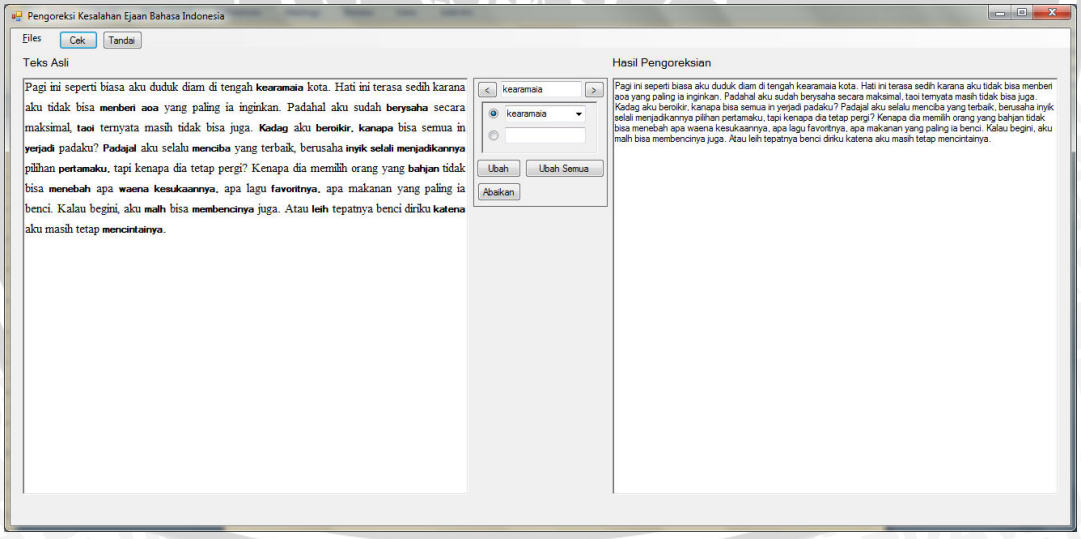
LAMPIRAN

Pengujian Pencarian Sugesti

Dalam pengujian pencarian sugesti, kata-kata yang digunakan terdapat pada paragraf yang ditunjukkan pada gambar berikut:

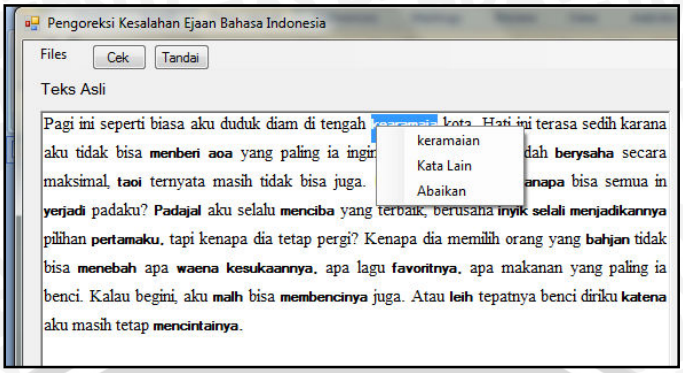
> Pagi ini seperti biasa aku duduk diam di tengah kearamaia kota. Hati ini terasa sedih karana aku tidak bisa menberi aoa yang paling ia inginkan. Padahal aku sudah berysaha secara maksimal, taoi ternyata masih tidak bisa juga. Kadag aku beroikir, kanapa bisa semua ini yerjadi padaku? Padajal aku selalu menciba yang terbaik, berusaha inyik selali menjadikannya pilihan pertamaku, tapi kenapa dia tetap pergi? Kenapa dia memilih orang yang bahjan tidak bisa menebah apa waena kesukaannya, apa lagu favoritnya, apa makanan yang paling ia benci. Kalau begini, aku malh bisa membencinya juga. Atau leih tepatnya benci diriku katena aku masih

Saat kata-kata tersebut dimasukkan ke dalam kolom teks masukan dan tombol cek ditekan, keluarannya adalah sebagai berikut:



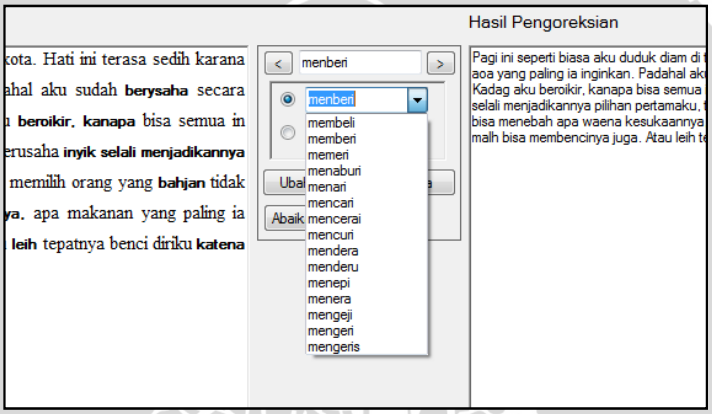
Hasil Pengecekan Ejaan

Kata-kata yang tidak tepat ejaannya akan tercetak tebal. Selanjutnya, sugesti bisa dicari dengan dua cara. Cara yang pertama adalah dengan melakukan klik kanan langsung pada kata yang salah, seperti ditunjukkan pada gambar berikut:



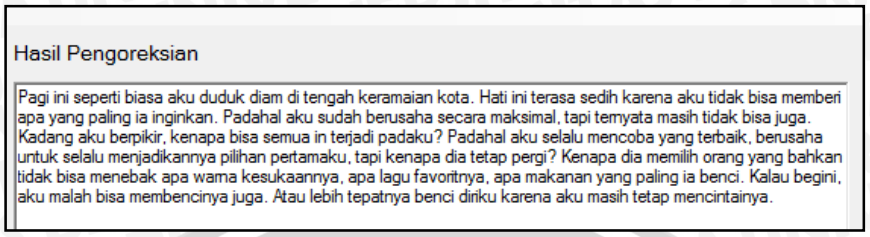
Memilih Sugesti

Cara yang kedua adalah dengan memilih pilihan sugesti yang ada pada *list* sugesti yang terdapat di sebelah kolom teks masukan, seperti ditunjukkan pada gambar berikut:



Memilih Sugesti

Setelah seluruh kata selesai dikoreksi, hasilnya terlihat di kolom hasil pengoreksian, seperti ditunjukkan pada gambar berikut:



Hasil Pengoreksian Kata