Sistem Koreksi Teks Otomatis berbasis Algoritma Program Dinamis

James Jaya (13511089)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
james.jaya@s.itb.ac.id

***Abstrak*—Permasalahan approximate string matching memiliki beberapa aplikasi salah satunya adalah sebagai pemeriksa ejaan. Pemeriksa ejaan dapat membenarkan ejaan secara otomatis atau memberikan saran kepada pengguna kata mana yang dimaksud. Penyelesaian approximate string matching dapat diselesaikan dengan program dinamis.**

***Kata kunci*— program dinamis, edit distance, string**

I. PENDAHULUAN

Permasalahan pencocokan *string* dapat dipilah menjadi dua bagian besar yaitu pencocokan *string* tepat (*exact string matching*) dan pencocokan *string* kira-kira (*approximate string matching*). *Exact string matching* adalah permasalahan pencocokan *string* dimana pola atau *pattern* yang dicari haruslah ditemukan sama persis pada *string*. Berlawanan dengan *exact string matching*, *approximate string matching* adalah permasalahan pencocokan *string* dimana pola atau *pattern* yang dicari tidak harus ditemukan sama persis pada *string*.

Permasalahan *exact string matching* dapat diselesaikan dengan berbagai algoritma. Beberapa algoritma yang dapat menyelesaikan permasalahan ini antara lain algorithma Knuth-Morris-Pratt (KMP) [1], algoritma Boyer-Moore [1], algoritma Bitap, algoritma Rabin-Karp, dan pencocokan secara naif juga dapat dilakukan [1]. Setiap algoritma tentu dengan kelebihan dan kekurangannya masing-masing. Misal algoritma Boyer-Moore klasik diketahui lebih cocok digunakan untuk *string* dengan jumlah alfabet yang banyak seperti *string* dari bahasa alami sedangkan algoritma KMP lebih cocok digunakan pada *string* dengan jumlah alfabet sedikit seperti pada pencocokan rantai DNA atau *string* biner [1].

Permasalahan *approximate string matching* tidak dapat diselesaikan dengan algoritma tersebut karena solusi yang diinginkan pada permasalahan ini bukanlah ada tidaknya sebuah pola dalam sebuah *string* melainkan himpunan pola yang dianggap paling mirip dengan *string*. Penyelesaian masalah ini dengan cara naif tentu akan sangat memakan waktu. Satu *string* harus dicocokkan dengan berbagai variasi dari pola AYAM. Misal terdapat *string* AYM akan dicocokan dengan pola AYAM, jika pendekatan naif dilakukan, perbandingan *string* yang dilakukan adalah *string* AYM dengan semua kemungkinan sub*string* dari AYAM. Solusi lain dari permasalahan ini adalah program dinamis.

Ada beberapa aplikasi dari permasalahan *approximate string matching* di dalam dunia nyata. Misal pencocokan rantai DNA dimana pola yang dicocokan tidak mungkin sama persis satu dengan yang lainnya. Apabila pola yang ditemukan semakin mirip maka dapat ditarik kesimpulan bahwa pola tersebut mungkin merepresentasikan kemiripan karakteristik atau kesamaan fungsi. Aplikasi lainnya adalah memeriksa kebenaran ejaan sebuah kata dan mengembalikan daftar kata yang paling mungkin menjadi kata yang sebenarnya. Aplikasi yang disebut terakhir tersebut saat ini banyak diimplementasikan pada modul pengirim pesan pada *smartphone* dan perangkat lunak *word processor*.

II. DASAR TEORI

A. Approximate String Matching

Permasalahan *approximate string matching* adalah permasalahan pencocokan *string* di mana pola dan *string* yang dibandingkan tidak perlu sama persis [4]. Terdapat sebuah nilai yang disebut *edit distance* yang menggambarkan seberapa mirip *string* dengan pola yang dibandingkan. Misal terdapat dua buah *string*, yaitu *string* A dan *string* B. *String* B dapat ditransformasikan menjadi *string* A dengan tiga jenis operasi: *deletion* (penghapusan), *insertion* (penyisipan), dan *substitution* (penggantian) [4]. Penamaan *string* A sebagai tujuan dari transformasi dari *string* B akan digunakan pada tulisan ini.

1) Operasi penghapusan
   Misal A = abcd dan B = aabbccdd. Maka *string* B dapat ditransformasikan menjadi *string* A dengan 4 buah operasi penghapusan, yaitu di posisi 2, 4, 6, dan 8.

$$\begin{array}{llcccccccc} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ A & = & a & - & b & - & c & - & d & - \\ B & = & a & \boxed{A} & b & \boxed{b} & c & \boxed{c} & d & \boxed{d} \end{array}$$

2) Operasi penyisipan
   Misal A = aabbccdd dan B = abcd. Maka *string* B dapat ditransformasikan menjadi *string* A dengan 4 buah operasi penyisipan yang dideskripsikan

pada gambar berikut.

$$\begin{array}{llllllllll} & & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ A & = & a & a & b & b & c & c & d & d \\ B & = & a & - & b & - & c & - & d & - \\ & & & a & & b & & c & & d \end{array}$$

3) Operasi penggantian
Misal A = abcd dan B = abce. Maka *string* B dapat ditransformasikan menjadi *string* A dengan sebuah operasi penggantian dengan mengganti karakter e menjadi karakter d.

$$\begin{array}{llllll} & & 1 & 2 & 3 & 4 \\ A & = & a & b & c & d \\ B & = & a & b & c & \boxed{e} \end{array}$$

Edit distance didefinisikan sebagai jumlah operasi minimum yang diperlukan untuk mentransformasi *string* B menjadi *string* A. *Edit distance* tersebut dinotasikan dengan ED(A, B). Semakin kecil *edit distance* antara A dan B maka semakin mirip *string* A dan B. *String* A dan *string* B dikatakan identik apabila ED(A, B) = 0. Penghitungan *edit distance* biasa dilakukan dengan menggunakan program dinamis.

Ada satu nilai lain yang dijadikan indikator kesamaan dua buah *string* yaitu Hamming *Distance*. Hamming *distance* sama dengan *edit distance* namun operasi yang diperhitungkan hanyalah operasi penggantian.

Contoh kasus perhitungan *edit distance* dan hamming *distance* adalah sebagai berikut. Misal A = abcd dan B = abcef, maka terdapat dua operasi minimum yang harus dilakukan untuk mentransformasikan *string* B menjadi *string* A yaitu penghapusan ‘f’ serta substitusi ‘e’ dengan ‘d’. ED(A, B) = 2 dan nilai Hamming *distance*nya adalah 1.

B. Program dinamis

Program dinamis menyelesaikan masalah dengan cara menggabungkan solusi dari submasalah [2]. Perbedaannya dengan pendekatan divide-and-conquer adalah divide-and-conquer membagi permasalahan menjadi submasalah secara rekursif lalu setiap submasalah tersebut digabungkan sehingga terbentuk solusi untuk keseluruhan permasalahan sedangkan program dinamis dapat dilakukan apabila setiap submasalah memiliki subsubmasalah yang sama [2]. Apabila submasalah memiliki subsubmasalah yang sebenarnya sama, maka subsubmasalah tersebut cukup diselesaikan sekali saja (pendekatan lain tetap mengkalkulasi solusi dari subsubmasalah yang sama berulang-ulang) dan solusi subsubmasalah tersebut disimpan ke dalam sebuah tabel [2]. Jika solusi dari subsubmasalah tersebut merupakan bagian dari submasalah lain maka komputasi subsubmasalah tersebut tidak dilakukan kembali dan solusi didapatkan dari tabel. Oleh karena itu, program dinamis juga disebut metode tabular [2].

Program dinamis biasa diaplikasikan ke dalam permasalahan optimasi. Permasalahan optimasi biasa memiliki banyak kemungkinan solusi. Setiap solusi memiliki sebuah nilai dan nilai dari solusi yang diharapkan biasanya nilai yang optimum (maksimum atau minimum).

Ada dua jenis program dinamis yaitu *top-down* dan *bottom-up [3]*. Program dinamis *top-down* mengisi tabel solusi saat dibutuhkan saja sedangkan program dinamis *bottom-up* mengisi tabel dari awal sampai akhir dimana pengisian nilai berikutnya memerlukan nilai-nilai sebelumnya [3]. Pendekatan *top-down* biasa digunakan melalui rekursi sedangnan *bottom-up* dilakukan dengan iterasi. Namun perbedaan kinerja diantara keduanya cukup kecil dan satu pendekatan mungkin lebih mudah dilakukan daripada pendekatan lain untuk permaslahan tertentu.

Top-Down	Bottom-Up
Kelebihan	
1. Lebih natural, kode mirip dengan solusi *complete* search rekursif. 2. Mengkomputasi submasalah yang dibutuhkan saja.	1. Lebih cepat apabila banyak submasalah yang dikunjungi berulang-ulang karena tidak ada overhead rekursi.

Terdapat beberapa contoh klasik permasalahan yang dapat diselesaikan dengan program dinamis, misalnya 0-1 *knapsack* dan coin change.

1. 0-1 *knapsack*
   Diberikan n buah objek, setiap objek memiliki nilai V_i dan bobot W_i di mana $0 \leq i \leq n-1$. Diketahui kapasitas dari *knapsack* adalah S. Pilihlah objek-objek yang akan dimasukkan ke dalam *knapsack* sehingga nilainya maksimum dengan syarat total bobot tidak melebihi S.
2. Coin change
   Uang kembali sebesar V akan diberikan ke seorang pelanggan. Diketahui himpunan denominasi uang yang jumlahnya n. Berapa jumlah minimum uang yang dapat digunakan untuk merepresentasikan V? Misal V = 10, n = 2 dimana tersedia denominasi {1, 5}. Terdapat tiga alternative solusi:
   a. 10 buah denominasi 1 (total 10 buah)
   b. 1 buah denominasi 5 dan 5 buah denominasi 1. (total 6 buah)
   c. 2 buah denominasi 5 (total 2 buah), ini adalah solusi optimum.

Banyak algoritma yang berbasis program dinamis, antara lain algoritma Cocke-Younger-Kasami (CYK) untuk memeriksa apakah sebuah *string* memenuhi aturan context-free grammar, algoritma Floyd-Warshall untuk mencari jarak terpendek dari semua node ke semua node lainnya dalam sebuah graf, dan algoritma Kadane untuk menghitung subarray dengan jumlah elemen maksimum.

III. Perancangan Sistem Koreksi Teks Otomatis

A. Rancangan Umum

Sistem koreksi teks otomatis ini berbentuk sebuah perangkat lunak berbasis *command-line interface* (CLI) diimplementasi dengan bahasa pemrograman C++. Program akan menerima dua jenis input yaitu file kamus yang berisikan kata-kata yang dijadikan sebagai rujukan pengujian ejaan dan jenis input lainnya adalah sebuah *string* yang akan diuji dan dicoba benarkan oleh program.

Awalnya, program akan membaca isi file kamus dan memasukkan setiap kata yang ada ke dalam memori. Kemudian, program akan membaca input *string* yang akan dikoreksi. *String* yang akan dikoreksi tersebut akan dipilah berdasarkan spasi yang berarti *string* akan dipotong per kata.

Setiap kata-kata tersebut, akan dibandingkan dengan kata-kata yang ada pada kamus. Ada dua kemungkinan aksi setelah ini yaitu:

1. Kata tersebut dianggap benar, apabila *edit distance* antara kata tersebut dengan salah satu kata yang ada pada kamus sama dengan 0.
2. Kata tersebut dianggap salah, karena tidak ada kata pada kamus yang memiliki *edit distance* bernilai 0 terhadap kata yang dibandingkan tersebut.

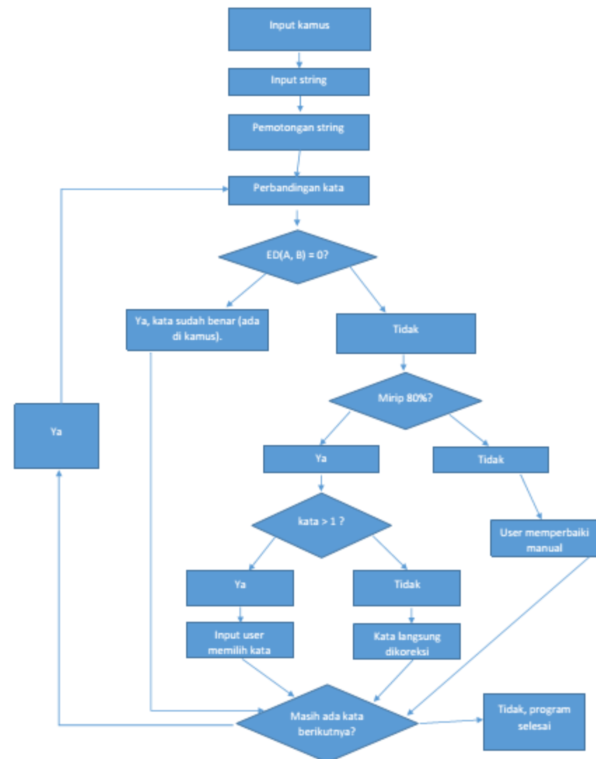
Apabila kata tersebut dianggap salah, ada dua kemungkinan aksi lainnya yang akan dilakukan program:

1. Apabila ditemukan kata yang 80% sama dengan kata yang dibandingkan, maka program akan mencoba memperbaiki kata tersebut.
2. Apabila tidak ditemukan kata yang 80% sama, program akan meminta konfirmasi dari pengguna apakah kata tersebut benar atau tidak, jika benar maka kata akan dimasukkan ke dalam kamus dan apabila tidak pengguna akan memperbaiki kata tersebut secara manual.

Selanjutnya, apabila ditemukan kata yang 80% sama dengan kata yang dibandingkan, program memiliki dua opsi:

1. Jika hanya ditemukan sebuah kata yang 80% sama dengan kata yang dibandingkan maka kata tersebut menjadi perbaikan atas kata yang dibandingkan.
2. Jika ditemukan lebih dari satu kata yang 80% sama dengan kata yang dibandingkan, pengguna akan diberikan pilihan untuk menentukan kata mana yang menjadi kata yang benar.

Keseluruhan proses ini akan terus diulang sampai semua kata dalam *string* telah diuji dan dianggap program sudah benar.



Gambar 1. Flow Chart Program

Keseluruhan proses ini akan terus diulang sampai semua kata dalam *string* telah diuji dan dianggap program sudah benar.

B. Perhitungan Edit Distance

Edit distance yang juga disebut Levenshtein *distance* dapat dihitung dengan berbagai pendekatan. Pertama, *edit distance* dapat dihitung dengan cara rekursif murni tanpa memoisasi. Cara ini tentu saja memakan waktu karena submasalah yang sama dihitung berulang-ulang. Kedua, *edit distance* dapat dihitung dengan program dinamis *bottom-up* yang iteratif.

Program dinamis menggunakan sebuah matriks sebesar panjang matriks A kali panjang matriks B untuk menyimpan *edit distance* antara semua prefiks *string* A dan semua prefiks *string* B. Fungsi perhitungan *edit distance* dideskripsikan pada gambar X.

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

Ketika salah satu panjang *string* adalah 0, maka *edit distance* kedua *string* tersebut adalah panjang *string* yang lainnya. Apabila tidak, *edit distance* adalah nilai minimum dari tiga nilai yaitu $lev(i-1,j)+1$ yang merupakan operasi penghapusan, $lev(i,j-1)+1$ yang merupakan operasi penyisipan, dan $lev(i-1,j-1)+1$ yang merupakan operasi penggantian. Hasil dari perhitungan setiap fungsi lev akan disimpan ke dalam matriks agar dapat digunakan pada perhitungan selanjutnya.

Berikut ini adalah pseudo-code fungsi yang digunakan untuk menghitung *edit distance*.

```
function ed(input a:string, b:string) :
integer {
   alen : integer; alen := (panjang
string a)
   blen : integer; blen := (panjang
string b)
   d : array of
integer[0..alen][0..blen]

   inisialisasi d dengan 0

   // pengisian kolom dan baris pertama
matriks
   i traversal [0..alen-1]
   begin
      d[i+1][0] := i + 1
   end

   j traversal [0..blen-1]
   begin
      d[0][j+1] := j + 1
   end

   j traversal [0..blen-1]
      begin
      i traversal [0..alen-1]
         begin
            if (a[i] = b[i]) then
            begin
               d[i+1][j+1] = d[i][j]
            end
            else
            begin
               d[i+1][j+1] =
min(min(d[i][j+1]+1, d[i+1][j]+1),
d[i][j]+1);
            end
         end
      end
   ➔ d[alen][blen]
```

Salah satu contoh penggunaan algoritma tersebut adalah sebagai berikut. Misal terdapat *string* A = innovation dan B = inovasisasi. Dibentuk matriks kosong berukuran 10 x 11 dimana 10 adalah panjang *string* A dan 11 adalah panjang *string* B. Awalnya, semua elemen matriks diinisialisasi menjadi 0. Matriks dianggap zero-based, penomoran baris dan kolom dimulai dari 0.

		i	n	o	v	a	s	i	s	a	s	i
	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0
v	0	0	0	0	0	0	0	0	0	0	0	0
a	0	0	0	0	0	0	0	0	0	0	0	0
t	0	0	0	0	0	0	0	0	0	0	0	0
i	0	0	0	0	0	0	0	0	0	0	0	0
o	0	0	0	0	0	0	0	0	0	0	0	0
n	0	0	0	0	0	0	0	0	0	0	0	0

Gambar 2. Matriks awal

Langkah berikutnya adalah pengisian baris dan kolom pertama dengan nomor baris dan nomor kolom. Hal ini merepresentasikan apabila satu karakter dari A dibandingkan dengan seluruh prefiks dari *string* B serta satu karakter dari B dibandingkan dengan seluruh prefiks dari *string* A. Hasil pengisian awal adalah matriks yang terlihat pada gambar 3.

		i	n	o	v	a	s	i	s	a	s	i
	0	1	2	3	4	5	6	7	8	9	10	11
i	1	0	0	0	0	0	0	0	0	0	0	0
n	2	0	0	0	0	0	0	0	0	0	0	0
n	3	0	0	0	0	0	0	0	0	0	0	0
o	4	0	0	0	0	0	0	0	0	0	0	0
v	5	0	0	0	0	0	0	0	0	0	0	0
a	6	0	0	0	0	0	0	0	0	0	0	0
t	7	0	0	0	0	0	0	0	0	0	0	0
i	8	0	0	0	0	0	0	0	0	0	0	0
o	9	0	0	0	0	0	0	0	0	0	0	0
n	10	0	0	0	0	0	0	0	0	0	0	0

Gambar 3. Matriks setelah pengisian awal

Kemudian pengisian dilakukan dari kolom ke-1 yang merupakan huruf pertama dari *string* B dibandingkan dengan setiap karakter dari *string* A sehingga pengisian dilakukan per kolom. Pengisian kolom ke-1 dapat dilihat pada gambar 4.

		i	n	o	v	a	s	i	s	a	s	i
	0	1	2	3	4	5	6	7	8	9	10	11
i	1	0	0	0	0	0	0	0	0	0	0	0
n	2	1	0	0	0	0	0	0	0	0	0	0
n	3	2	0	0	0	0	0	0	0	0	0	0
o	4	3	0	0	0	0	0	0	0	0	0	0
v	5	4	0	0	0	0	0	0	0	0	0	0
a	6	5	0	0	0	0	0	0	0	0	0	0
t	7	6	0	0	0	0	0	0	0	0	0	0
i	8	7	0	0	0	0	0	0	0	0	0	0
o	9	8	0	0	0	0	0	0	0	0	0	0
n	10	9	0	0	0	0	0	0	0	0	0	0

Gambar 4. Matriks dengan kolom ke-1 terisi

Setiap kolom menggambarkan prefiks dari *string* B dan setiap baris menggambarkan prefiks dari *string* A. Sebagai contoh kolom ke-3 dan baris ke-4 membandingkan sub*string* ino dan inno, maka sel tersebut akan bernilai 1 karena hanya dibutuhkan 1 operasi untuk mentransformasi ino menjadi inno. Pengisian matriks dimulai dari prefiks *string* B terkecil dibandingkan dengan setiap prefiks *string* A sampai *string* B dibandingkan secara keseluruhan.

		i	n	o	v	a	s	i	s	a	s	i
	0	1	2	3	4	5	6	7	8	9	10	11
i	1	0	1	2	3	4	5	6	7	8	9	10
n	2	1	0	1	2	3	4	5	6	7	8	9
n	3	2	1	1	2	3	4	5	6	7	8	9
o	4	3	2	1	2	3	4	5	6	7	8	9
v	5	4	3	2	1	2	3	4	5	6	7	8
a	6	5	4	3	2	1	2	3	4	5	6	7
t	7	6	5	4	3	2	2	3	4	5	6	7
i	8	7	6	5	4	3	3	2	3	4	5	6
o	9	8	7	6	5	4	4	3	3	4	5	6
n	10	9	8	7	6	5	5	4	4	4	5	6

Gambar 5. Matriks telah terisi penuh

Nilai *edit distance* antara *string* A dan *string* B adalah sel matriks yang terkanan bawah dalam kasus ini *edit distance*nya adalah 6.

C. Pemilihan Kata yang Benar

Proses pemilihan kata yang benar dilakukan dengan cara mencocokan kata yang diuji dengan semua kata yang ada di dalam kamus. Setiap perbandingan kata dengan kata akan menghasilkan nilai *edit distance*, kemudian hasil tersebut akan disorting berdasarkan *edit distance* kemudian aplikasi secara otomatis akan memilih kata dengan *edit distance* yang terkecil.

Sebagai contoh misal kata yang akan diuji adalah MAGNEX dan pada kamus terdapat 10 kata yaitu {MAGLEV, MINUM, MAGNET, MANGAN, MAAG, GAME, GAMA, MAKAN, MAGNESIUM, AGNI}. Hasil perbandingan akan menghasilkan nilai yang tertulis pada tabel berikut.

String	Edit *Distance*
MAGLEV	2
MINUM	4
MAGNET	1
MANGAN	4
MAAG	4
GAME	4
GAMA	5
MAKAN	4
MAGNESIUM	4
AGNI	3

Perangkat lunak akan langsung mengkoreksi kata MAGNEX menjadi MAGNET karena tidak ada kandidat lain yang memiliki *edit distance* terkecil.

Contoh kasus lain misal kata yang akan diuji adalah MAGNUM. Perbandingan dengan menggunakan kamus yang digunakan pada contoh sebelumnya, hasilnya dapat dilihat pada tabel berikut.

String	Edit *Distance*
MAGLEV	3
MINUM	2
MAGNET	2
MANGAN	4
MAAG	4
GAME	5
GAMA	5
MAKAN	4
MAGNESIUM	3
AGNI	3

Ada dua alternatif kata dalam kamus yang memiliki *edit distance* minimum yaitu MINUM dan MAGNET, maka pengguna akan diberikan pilihan manakah kata yang benar MINUM atau MAGNET.

IV. PENGUJIAN

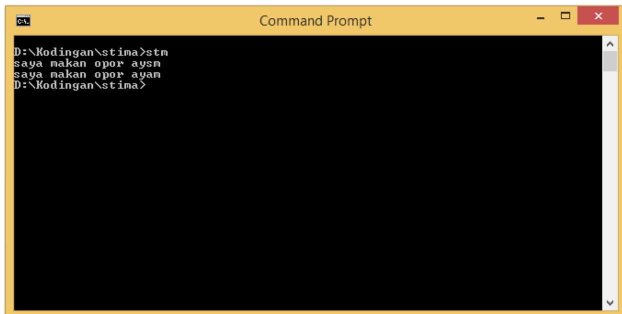
Pengujian akan dilakukan dengan dua buah *string* bahasa Indonesia dan sebuah *string* dalam bahasa Inggris. File kamus bahasa Indonesia yang digunakan berasal dari pranala berikut http://wa2010.ee.itb.ac.id/files/daftar-kata-bahasa-indonesia.txt. *String* mengandung beberapa kata yang benar, definisi kata yang benar adalah kata-kata yang ada di dalam kamus.

Percobaan pertama menggunakan *string* bahasa Indonesia. *String* yang akan diujikan adalah *string* yang tertulis pada kotak berikut.

```
saya makan opor aysm
```

String tersebut merupakan salah satu kesalahan yang mungkin cukup umum terjadi karena penulisan kata ayam menjadi aysm karena huruf s berdekatan dengan huruf a pada keyboard QWERTY. *String* tersebut kemudian dimasukkan ke dalam aplikasi. Program kemudian membenarkan *string* tersebut menjadi *string* berikut. Perubahan langsung dilakukan karena tidak ada kandidat kata lain yang mungkin mengisi kesalahan tersebut.

saya makan opor ayam



Gambar 6. Hasil pengujian pertama

Percobaan kedua adalah percobaan dengan *string* di mana program akan memberikan sejumlah pilihan kata yang benar karena terdapat beberapa kandidat yang memiliki kemiripan yang sama dengan kata yang diuji.

kamu minum aer dan makan sapi lade hitem

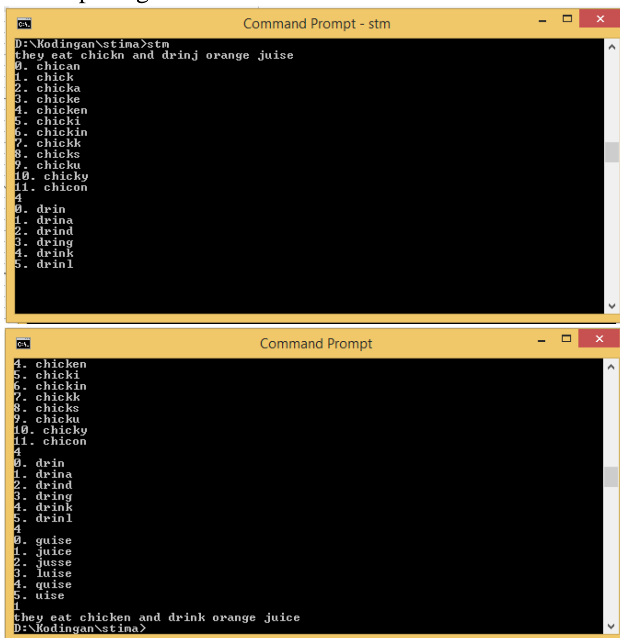
String tersebut memiliki tiga kesalahan yaitu aer, lade, dan hitem. Ketika dimasukkan ke aplikasi, aplikasi mendeteksi ketiganya dan memberikan saran perbaikannya serta menunggu user memasukkan input. Hasil eksekusi program terlihat pada gambar 7.

Gambar 7. Hasil pengujian kedua

String selanjutnya yang akan diuji adalah *string* dalam bahasa Inggris. Kamus yang digunakan berasal dari pranala berikut http://invokeit.wordpress.com/frequency-word-lists/. Kamus tersebut juga mengandung frekuensi seberapa sering kata tersebut muncul dalam bahasa Inggris. *String* yang akan diujikan adalah *string* berikut.

they eat chickn and drinj orange juise

Program menemukan tiga kesalahan dan tiga kali menawarkan kata-kata perbaikan. Eksekusi program dapat dilihat pada gambar 8.



Gambar 8. Hasil pengujian ketiga

they eat chicken and drink orange juice

String tersebut dibenarkan menjadi *string* berikut dengan memberikan opsi-opsi pilihan kepada user.

VII. PENGHARGAAN

Penulis ingin mengucapkan syukur kepada Tuhan Yang Maha Kuasa atas segala kemampuan yang diberikan sehingga makalah ini selesai tepat waktu. Terima kasih kepada kedua orang tua saya yang selalu mendoakan yang terbaik untuk saya. Penulis juga mengucapkan terima kasih kepada dosen mata kuliah IF2211 Strategi Algoritma semester ganjil 2013/2014, Dr. Ir. Rinaldi Munir, MT. dan Dr. Masayu Leylia Khodra, ST., MT. atas segala ilmu dan bimbingan yang telah diberikan. Terakhir, terima kasih kepada teman-teman peserta kuliah IF2211 yang membuat kuliah ini menyenangkan.

REFERENSI

[1] Gossett, Eric. *Discrete Mathematics with Proof.* John Wiley and Sons, 2009.
[2] Cormen, Thomas H.; Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. *Introduction to Algorithms* (3rd ed.). MIT Press and McGraw-Hill, 2008.
[3] Halim, Steven and Felix Halim. Competitive Programming (3rd ed). Lulu Publishing, 2012.
[4] http://www8.cs.umu.se/kurser/TDBAfl/VT06/algorithms/BOOK/BOOK2/NODE46.HTM

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Desember 2013

James Jaya (13511089)