ISSN 2355-3286

Optimasi Pencarian Kata pada Aplikasi Penerjemah Bahasa Mandarin – Indonesia Berbasis Android dengan Algoritma Levenshtein Distance

Peggy, Seng Hansun
Program Studi Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, Indonesia
phe_gy@yahoo.com, hansun@umn.ac.id

Diterima 3 Maret 2015
Disetujui 5 Mei 2015

Abstract—In the Chinese-Indonesian language translator application on electronic device, such as Android, input is all about pinyin. Sometimes, some errors could occured. User can not find the word which he is searching for. These errors can increase the number of processing time. A solution to solve these errors is performing optimization on the application, i.e. by providing some alternative approaches word which typed by user. Application will sort alternative words according to the smallest distance to the largest distance. Then, application will display alternative words order from top priority which has the smallest distance, to low priority which has the largest distance. In brief, translator application with optimization is useful to shorten processing time of translating from Chinese to Indonesian by providing alternative words. Optimization is done with Levenshtein Distance algorithm. Levenshtein Distance algorithm will calculate the difference between the distance of the word user typed among alternative words.

Index Terms—Android, Chinese, Indonesian, Levenshtein Distance, optimization, translator application

I. PENDAHULUAN

Bahasa adalah alat komunikasi, baik secara verbal maupun non-verbal. Pada abad ke-21 ini, masyarakat mengenal adanya bahasa universal. Bahasa universal adalah bahasa pokok yang digunakan hampir di seluruh belahan dunia untuk berkomunikasi. Salah satunya adalah Bahasa Mandarin. Bahasa Mandarin merupakan bahasa universal kedua setelah bahasa Inggris. Bahasa yang merupakan bahasa ibu negara Cina ini banyak dipakai oleh masyarakat Indonesia khususnya etnis keturunan. Namun, seiring berjalannya waktu Bahasa Mandarin kian penting, sehingga banyak sekolah yang memasukannya ke dalam pokok pelajaran, baik mata pelajaran wajib maupun ekstrakulikuler. Bahkan, sudah ada sekolah yang menetapkan sekolahnya sebagai sekolah *tri-lingual*, yaitu sekolah yang menggunakan tiga bahasa, yaitu Bahasa Indonesia, Bahasa Inggris, dan Bahasa Mandarin. Belajar Bahasa Mandarin tidak sulit jika dilandasi kemauan yang keras. Komunikasi dalam Bahasa Mandarin pada *level* aktif dapat memperbesar peluang untuk sukses, terutama bagi yang ingin melanjutkan studi atau yang ingin sekedar bekerja di negeri tirai bambu.

Pengucapan kata dalam bahasa mandarin seringkali mengundang tanya tentang apa arti dari kata yang terucap. Misalnya, ketika seseorang mengatakan *peng you* (朋友; *pinyin*: *péng yǒu*). Kata *péng yǒu* yang berarti teman atau sahabat dapat diterka penulisan *pinyin* melalui tekanan bunyi pada kata *péng* dan *yǒu*. *Pinyin* adalah bentuk fonetik dari bahasa mandarin dengan aturan-aturan yang telah ditetapkan seperti penulisan tekanan bunyi pengucapan dengan tujuan untuk mempermudah proses *input* ke media elektronik [1]. Pada aplikasi penerjemahan di media elektronik, terkadang media itu sendiri tidak mendukung penulisan, baik *pinyin* maupun karakter Cina. Untuk itu dalam penelitian ini penulis mengembangkan sebuah aplikasi yang dapat memberikan terjemahan berdasarkan *input* berupa *pinyin,* dengan atau tanpa mengikutsertakan tekanan bunyi.

Kesalahan dalam penulisan *pinyin* sering kali terjadi dan menambah waktu proses pencarian. Untuk itu, agar mengurangi peluang terjadinya kesalahan kembali, dilakukan optimasi pada aplikasi penerjemahan ini. Optimasi dalam aplikasi ini menggunakan algoritma *Levenshtein Distance.* Algoritma ini akan membandingkan beberapa kata yang ditemukan dan menghitung jarak perbedaan pada kata *input* dan kata yang ditemukan. Kemudian, aplikasi ini akan memberikan beberapa alternatif kata yang telah diurutkan menurut perbedaan selisih yang paling sedikit. Sebelumnya telah ada yang menggunakan algoritma Levenshtein Distance untuk membangun aplikasi pencarian kata [2] dan pengolahan *string suggestion* [3]. Sedangkan aplikasi kamus penerjemahan, sebelumnya telah berhasil dibangun dengan menggunakan bahasa Aceh – Indonesia berbasis Java untuk Mobile [4].

Berdasarkan penelitian yang sudah ada sebelumnya, aplikasi ini menggunakan algoritma Levenshtein Distance untuk mengoreksi kesalahan kata yang di-*input* oleh *user*. Pada kesempatan ini penulis akan menggunakan Bahasa Mandarin dengan *input pinyin* sebagai penelitian.

Penelitian ini memprioritaskan kemudahan seseorang dalam menemukan kata terjemahan Bahasa Mandarin menjadi Bahasa Indonesia. *Output* dari aplikasi ini adalah karakter Cina, *pinyin,* dan arti terjemahan. Penelitian ini berbasis *mobile* dengan sistem operasi Android dengan basis data SQLite dengan pertimbangan Android adalah salah satu sistem operasi yang paling banyak terjual pada pasar *smartphone* dan *tablet* yang digunakan saat ini [5].

II. ALGORITMA LEVENSHTEIN DISTANCE

Algoritma *Levenshtein Distance* atau *Edit Distance* adalah algoritma pencarian jumlah perbedaan *string* yang ditemukan oleh Vladimir Levenshtein, seorang ilmuwan Rusia, pada tahun 1965 [6]. Algoritma ini banyak digunakan dalam berbagai bidang, misalnya pencarian *string,* pendeteksi plagiarisme dan *speech recognition*. Algoritma ini merupakan perkembangan dari *dynamic programming* dimana algoritma ini melakukan perhitungan dengan menggunakan matriks pembanding dan memberi *output* jumlah perbedaan di antara dua *string* yang disebut dengan *distance.* Algoritma ini menentukan *distance* berdasarkan jumlah minimum perubahan yang terjadi ketika terjadi transformasi dari bentuk *string* awal ke bentuk *string* lain. Dalam algoritma Levenshtein Distance terdapat 3 macam operasi yang digunakan [7], yaitu

1. Insertion

Insertion adalah operasi melakukan penyisipan sebuah karakter ke dalam sebuah *string* tertentu.

2. Deletion

Deletion adalah operasi melakukan penghapusan sebuah karakter di dalam sebuah *string.*

3. Substitution

Substitution adalah operasi penggantian pada sebuah karakter pada posisi tertentu dengan karakter lain.

Algoritma Levenshtein Distance selain untuk menghitung *distance,* juga digunakan untuk menghitung jumlah operasi minimum yang diperlukan untuk mengubah *string* pertama dengan *string* kedua. Berikut ini adalah algoritma Levenshtein Distance yang digunakan untuk mencari nilai *distance* antar kedua *string.* Diketahui bahwa nilai *m* adalah panjang dari *string* pertama dan *n* adalah panjang dari *string* kedua.

```
int computeDistance(String str1, String str2)
{
declare m = length of str1
declare n = length of str2

declare costs[n+1]

FOR i from 0 to m
        DECLARE lastValue = i
        FOR j from 0 to n
                IF i = 0 THEN
                        costs[j] = j
                ELSE
                        IF j IS GREATER THAN 0 THEN
                                DECLARE newValue = costs[j - 1]
                                IF str1.charAt(i - 1) IS NOT EQUAL TO
                                   str2.charAt(j - 1) THEN
                                   newValue = minimum(newValue,
                                                      lastValue,
                                                      costs[j]) + 1
                                END IF
                                costs[j - 1] = lastValue
                                lastValue = newValue
                        END IF
                END IF
        END FOR
        IF i IS GREATER THAN 0 THEN
                costs[n] = lastValue
        END IF
END FOR

RETURN costs[n]
}
```

Gambar 1. *Pseudocode* Levenshtein Distance

Dari *pseudocode* yang dijabarkan pada gambar 1 diterangkan bahwa algoritma Levenshtein Distance menyimpan data dalam bentuk *array* yang berukuran *n* + 1 yang diberi nama *costs*. Dimana variabel *m* adalah panjang karakter *string* pertama dan *n* adalah panjang karakter *string* kedua. Selanjutnya, iterasi dilakukan sebanyak *m* dengan terdapat iterasi sebanyak *n* di setiap iterasi *m.* Pada setiap iterasi *m*, nilai *lastValue* dideklarasikan sama dengan nilai *i.* Kemudian, pada iterasi *n,* terjadi pengecekan apakah *i* bernilai 0. Jika tidak, maka akan terjadi pengecekan apakah nilai *j* pada *n* lebih besar dari angka 0. Apabila bernilai *TRUE,* maka nilai variabel *newValue* akan diberikan nilai *costs* pada *j* kurang dari 1. Pada tahap berikutnya, terjadi pengecekan kesamaan huruf antar karakter. Jika kondisi terpenuhi, maka nilai variabel *newValue* akan diberikan nilai minimum dari *newValue, lastValue,* atau *costs* pada *index j.* Nilai minimum yang sudah didapat akan ditambahkan 1. Hasil akhir dari iterasi ini berupa sebuah *array* yang telah memiliki nilai pada setiap *index*-nya. Algoritma ini akan mengembalikan nilai pada *array costs* yang berada pada posisi *index* ke-*n.* Nilai ini adalah nilai *distance* antara *string* pertama dengan *string* kedua.

Contoh kasusnya, yaitu diantara dua *string, yaitu* "kitten" dan "sitting". Kedua kata ini mempunyai nilai *distance* sebesar 3. Berikut ini adalah operasi-operasi yang dilakukan untuk mengubah kata "kitten" menjadi kata "sitting.

1. Kitten > Sitten, terjadi subsitusi atau penggantian karakter dari “K” menjadi “S”.
2. Sitten > Sittin, terjadi substitusi pada karakter “e” menjadi “i”.
3. Sittin > Sitting, terjadi penyisipan karakter “g” di belakang kata “sittin”.

Algoritma Levenshtein Distance dapat dihitung dengan menggunakan matriks perbandingan. Berikut ini adalah tabel matriks perhitungan yang dihasilkan untuk mencari nilai *distance* dari kata “kitten” dan “sitting”.

Tabel 1. Matriks perhitungan Levenshtein Distance

		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

Tabel ini menunjukkan bahwa kata “kitten” dan “sitting” mempunyai nilai *distance* sebesar 3. Dimana untuk setiap iterasi dicari operasi yang paling minimum yang bisa digunakan diantara operasi *insertion, deletion,* dan *substitution.*

III. METODOLOGI PENELITIAN

Sistematika metodologi penelitian yang dilakukan adalah sebagai berikut.

1. Studi Literatur

Dilakukan studi terhadap kebutuhan dan referensi yang dibutuhkan untuk penelitian.

2. Perancangan aplikasi

Berisi perancangan kebutuhan sistem dan desain keseluruhan sistem.

3. Pengujian dan Pembahasan

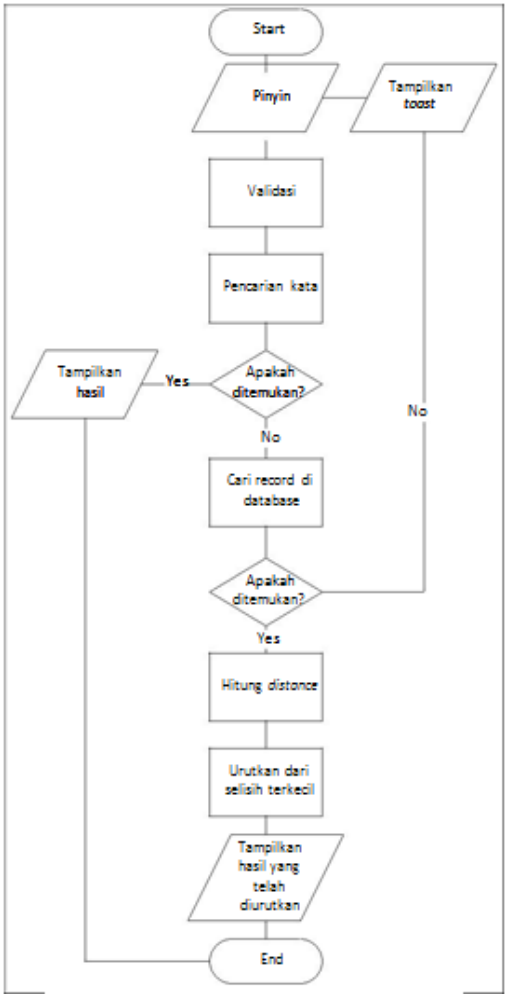
Berisi penjelasan mengenai implementasi dan hasil uji coba sistem

4. Simpulan dan Saran

Berisi kesimpulan penelitian dan saran untuk penelitian selanjutnya.

IV. PENERAPAN ALGORITMA LEVENSHTEIN

Dalam pengembangan dengan menggunakan sistem operasi Android, setiap aktifitas dilakukan di dalam *activity. Activity* ini pada awalnya berisi sebuah *text field* dan sebuah *button search. Activity* ini meminta sebuah *input* yang kemudian dijadikan parameter untuk proses pencarian. Tombol *search* adalah tombol yang digunakan untuk memulai proses pencarian. Apabila pencarian dengan parameter ini berhasil ditemukan, maka detail kata akan ditampilkan pada *activity* ini. Detail yang ditampilkan adalah karakter Mandarin, *pinyin,* arti kata, dan sebuah tombol bergambar pengeras suara, dan tombol untuk melanjutkan pencarian. Apabila tombol bergambar pengeras suara ditekan, maka aplikasi ini akan memutar *audio* dari kata yang ditemukan. Tombol untuk melanjutkan pencarian apabila ditekan maka akan berpindah ke *activity* daftar kata alternatif yang telah disusun berdasarkan algoritma untuk optimasi pencarian kata. Berikut ini adalah *flowchart* penerjemahan kata dengan penerapan algoritma Levenshtein Distance.



Gambar 2. *Flowchart* aplikasi penerjemah

Chinese character dapat di-*input* dengan

menggunakan *encoding* utf-8. Pada pengembangan dengan Android, setiap *activity* yang digunakan bila terdapat teks dalam bahasa dengan guratan-guratan seperti *chinese character*, secara otomatis akan terintegrasi dengan *encoding* yang sesuai. Data yang digunakan diperoleh dari sebuah situs pembelajaran dengan tidak memungut biaya [8]. Berikut ini merupakan dokumentasi untuk pemberian kata alternatif dengan menggunakan algoritma Levenshtein Distance untuk mencegah kesalahan *input* berulang-ulang.

Gambar 3. Tampilan daftar kata alternatif

Pada kondisi dimana *input* sebagai parameter tidak ditemukan dalam basis data maka yang ditampilkan adalah tombol untuk pencarian lebih lanjut. Tombol ini apabila ditekan maka akan berpindah ke *activity* yang berisi daftar kata alternatif yang disusun berdasarkan algoritma Levenshtein Distance untuk optimasi pencarian kata. Kemudian, pada kondisi dimana tidak terdapat *input* dan tombol *search* ditekan, proses penerjemahan kata tidak dapat dilaksanakan. Pada *activity* ini juga terdapat sebuah tombol yang diberi nama "Bookmark this word". Tombol ini berfungsi untuk menambahkan kata ke dalam daftar penyimpanan.

Gambar 4. Tampilan fitur *bookmarks*

V. SIMPULAN DAN SARAN

Berdasarkan hasil implementasi dan uji coba yang dilakukan sebelumnya, simpulan dari penelitian ini adalah penerapan algoritma Levenshtein Distance dalam pencarian kata pada aplikasi yang dapat menerjemahkan Bahasa Mandarin ke dalam Bahasa Indonesia berhasil dilakukan. Penerapan algoritma Levenshtein Distance dapat memberikan kata alternatif terhadap kesalahan *input* yang dilakukan oleh *user.*Dengan demikian, *user* tidak perlu secara terus menerus melakukan kesalahan dalam hal pencarian terjemahan.

Dari penelitian yang telah dilakukan, masih terdapat kekurangan-kekurangan antara lain.

1. Aplikasi penerjemahan seperti ini sebaiknya tidak hanya menerjemahkan kata dari Bahasa Mandarin menjadi Bahasa Indonesia tetapi juga sebaliknya, menerjemahkan kata dari Bahasa Indonesia menjadi Bahasa Mandarin.

2. Data yang disimpan dalam *database* ditambah perbendaharaan katanya untuk beberapa kategori, misalnya kata-kata yang berhubungan dengan teknologi, profesi, dan hal-hal yang menjadi *trend* saat ini.

3. Untuk menambah performa dari aplikasi *translator,* disarankan *input* dari *user* tidak hanya berupa *pinyin,* tetapi juga pencarian berdasarkan karakter Bahasa Mandarin seperti aplikasi penerjemah lainnya.

4. Optimasi pencarian kata juga dapat lebih dimaksimalkan dengan mengunakan algoritma yang lebih baik dari algoritma Levenshtein Distance yaitu algoritma Trie. Algoritma Damerau – Levenshtein Distance yang merupakan pengembangan dari algoritma Levenshtein Distance yang memungkinkan terjadinya operasi transposisi juga dapat digunakan untuk memaksimalkan pencarian.

DAFTAR PUSTAKA

[1] Lee, Jennifer. *Where the PC is Mightier Than the Pen*. Dalam http://www.nytimes.com/2001/02/01/technolog y/where-the-pc-is-mightier-than-the-pen.html yang diakses pada tanggal 20 Oktober 2013.

[2] Ilmy, Muhamad Bahari, dkk. 2006. Dalam jurnal dengan judul *Penerapan String Suggestion dengan Algoritma Levenshtein Distance dan Alternatif Algoritma Lain dalam Aplikasi*. Bandung: Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung.

[3] Adiwidya, A. 2009. *Algoritma Levenshtein dalam Pendekatan Approximate String Matching*. Dalam http://informatika.stei.itb.ac.id yang diakses pada tanggal 01 Oktober 2013.

[4] Mutiawani, V., Juwita, Irvanizam. 2009. Dalam jurnal dengan judul *Aplikasi Kamus Dwibahasa Aceh – Indonesia Berbasis Java untuk Telepon Genggam*. Aceh: Universitas Syuah Kuala.

[5] Gartner, Inc. 2013. *Gartner Says Smartphone Sales Grew 46.5 Percent in Second Quarter of 2013 and Exceeded Feature Phone Sales for First Time*. Dalam http://www.gartner.com/newsroom/id/2573415 yang diakses pada tanggal 30 September 2013

[6] Levenshtein, Vladimir. 1965. *Binary Codes Capable of Correcting Deletions, Insertions, and Reversal*. Russia: Soviet Physics Doklady.

[7] Rosetta Code. 2011. Dalam http://rosettacode.org/wiki/Levenshtein_distance yang diakses pada tanggal 10 Desember 2013.

[8] Shea, Marilyn. 2006. Dalam http://hua.umf.maine.edu/China/database.html. Farmington: University of Maine.