SISTEM PENGKOREKSIAN KATA KUNCI DENGAN MENGGUNAKAN METODE *LEVENSHTEIN DISTANCE*

Studi Kasus Pada Website Universitas Halmahera

Oleh :
Benisius

Sejumlah penelitian terhadap mesin pencari *(search engine)* menyimpulkan bahwa rata-rata kesalahan pengejaan kata kunci yang dilakukan pengguna cukup tinggi. Oleh karenanya diperlukan suatu sistem yang dapat melakukan pengkoreksian kata kunci pada aplikasi pencarian kata berbasis web. Penelitian ini bertujuan mengimplementasikan salah satu metode *approximate string matching*, yakni *Levenshtein Distance* dalam mengatasi permasalahan tersebut. Pembahasan mengenai proses penghitungan jarak antar string, proses kerja mesin pencari berbasis web yang meliputi proses *crawling*, *indexing* sampai dengan menampilkan hasil pencarian beserta saran kata kunci adalah fokus pada penelitian ini. Pengujian dilakukan dengan menggunakan beberapa jenis inputan kata kunci serta mengamati bagaimana hasil pencarian dan saran kata kunci yang dihasilkan sistem diperoleh.

Keywords: *Levenshtein Distance*, *approximate string matching*, *mesin pencari, pengkoreksian kata kunci, crawling, indexing.*

A. Pendahuluan

Fasilitas pencarian kata sudah berkembang dengan sangat cepat. Jika semula hasil pencarian hanya dilakukan berdasarkan kata kunci yang diinputkan oleh pengguna, dalam artian apabila pengguna menginputkan kata kunci secara tidak tepat maka hasil pencarian yang dikembalikan tidaklah sesuai, maka saat ini sudah dimungkinkan untuk melakukan pengkoreksian kata kunci berdasarkan inputan pengguna.

Metode Levenshtein Distance

Metode *Levenshtein Distance* dapat digunakan untuk melakukan perhitungan beda jarak antara dua string. Jarak atau *distance* adalah jumlah minimum dari operasi hapus, insert atau substitusi yang dibutuhkan untuk merubah string asal (s) menjadi string target (t). Sebagai contoh:

- Jika s adalah "coba" dan t adalah "coba", maka LD(s,t) = 0, dikarenakan tidak ada transformasi yang dibutuhkan. Kedua string adalah identik.
- Jika s adalah "coba" dan t adalah "coka", maka LD(s,t) = 1, dikarenakan satu substitusi (merubah "b" ke "k") dicukupkan untuk mentransform s menjadi t.

Perhitungan harga pengeditan pada setiap operasi yang dilakukan adalah sesuai dengan aturan berikut:

- d(a,ε) = 1 harga untuk menghapus substring a
- d(ε,a) = 1 harga untuk penyisipan substring a

– d(a,b) = 1 harga untuk substitusi substring a ke substring b
– d(a,a) = 0

Semakin besar angka yang dihasilkan oleh operasi *Levenshtein Distance* maka semakin besar perbedaan di antara kedua string tersebut.

Algoritma *Levenshtein Distance*

1. m = panjang kata1 (kata kunci yang diinputkan oleh pengguna).
2. n = panjang kata2 (kata dari tabel kata yang digunakan untuk pembanding).
3. d[0,0] = 0.
4. Untuk i = 1 sampai m, kerjakan d[i,0] = d[i-1,0] + c(kata1$_i$, ε).
5. Untuk j = 1 sampai n, kerjakan d[0,j] := d[0,j-1] + c(ε, kata2$_j$).
6. Menghitung nilai array dalam matrik
   Untuk i = 1 sampai m, kerjakan
   Untuk j = 1 sampai n, kerjakan
   d[i,j] = Min { d[i-1,j] + c(kata1$_i$, ε); d[i,j-1] + c(ε, kata2$_j$); d[i-1,j-1] + c(kata1$_i$, kata2$_j$) }

Hasil perhitungan jarak ditunjukkan pada kolom d[n,m].

Cara Penelitian

Penelitian ini merancang sistem pengkoreksian kata kunci dengan mengaplikasikan metode *Levenshtein Distance*. Sistem ini bertujuan membantu pengguna dalam mengatasi permasalahan yang sering timbul pada saat memanfaatkan fasilitas pencarian pada website yakni hasil pencarian yang tidak optimal dikarenakan terjadi kesalahan penginputan kata kunci. Apabila hal itu terjadi maka sistem diharapkan dapat melakukan pengkoreksian untuk lalu memberikan saran kata kunci yang dapat dipakai untuk melakukan proses pencarian berikutnya. Saran kata kunci diperoleh dengan cara membandingkan kata kunci yang dianggap salah dengan kata-kata terindeks yang berasal dari website. Hasil perbandingan kata yang memiliki nilai jarak terkecil akan terpilih sebagai saran kata kunci.

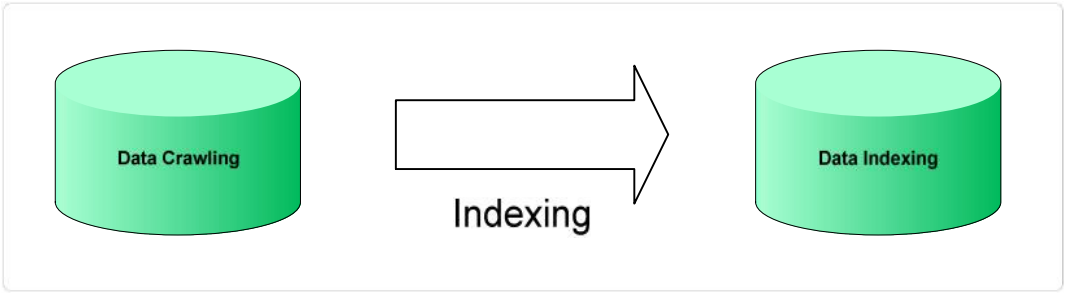
Pengindeksan Kata

Kata-kata yang akan digunakan sebagai kata saran merupakan kata-kata terindeks yang berasal website www.uniera.ac.id. Kata-kata ini diperoleh melalui dua tahapan yakni crawling dan indexing.

1. **Crawling**, proses ini bertujuan untuk menyediakan informasi konten atau keseluruan isi halaman yang terdapat pada website www.uniera.ac.id ke dalam table di *database.* Melalui proses *crawling* sistem akan menerima data-data sebagai berikut:
   – **Url:** merupakan alamat url dari konten yang ter-*crawling*.
   – **Header:** status HTTP yang diterima dari konten ter-*crawling*.
   – **Title:** judul dari setiap konten ter-*crawling* yang diambil dari kata-kata yang diapit oleh tag <title> … </title>.
   – **Content:** isi dari konten yang ter-*crawling.*

Gambar 4.5 Ilustrasi proses *crawling*

2. **Indexing**, proses ini pada akhirnya akan menghasilkan suatu bank kata yang perbendaharaan katanya berasal dari content hasil crawling. Kata-kata inilah yang kemudian akan dibandingkan dengan kata kunci yang diinputkan oleh pengguna untuk dapat menghasilkan saran kata kunci bagi pengguna dalam melakukan proses pencarian selanjutnya secara lebih akurat.



Gambar 4.7 Ilustrasi proses *indexing*

Proses Pencarian dan Pengkoreksian Kata Kunci

Proses pencarian kata adalah proses yang dilakukan tepat setelah pengguna memberikan inputan berupa kata kunci dan menekan tombol cari.

Dalam suatu pencarian, sistem akan senantiasa mengembalikan dua hasil yakni pencarian berhasil atau gagal. Suatu pencarian dikatakan berhasil apabila seluruh kata kunci berhasil mengenai target sedangkan pencarian gagal adalah kondisi sebaliknya.

Sistem pengkoreksian kata kunci akan bekerja pada saat sistem mengembalikan hasil pencarian gagal. Pada saat ini sistem akan melakukan proses saranKata.

Proses penghitungan jarak dengan metode *Levenshtein Distance*

Proses pemilihan saran kata kunci akan dilakukan setiap kali sistem mengembalikan nilai *false* yang menandakan pencarian terhadap kata

kunci tersebut tidak berhasil ditemukan. Pada proses ini pertama-tama sistem akan menfilter semua kata dari tabel KATA yang memiliki panjang antara (panjang kata kunci – 3) s.d. (kata kunci + 3). Pembatasan ini adalah batas aman yang ditetapkan dan dianggap sudah cukup untuk mendeteksi kesalahan ejaan yang dilakukan pengguna selain juga untuk mempersingkat waktu eksekusi program dimana sistem tidak perlu melakukan pengecekan terhadap semua kata yang ada.

Selanjutnya, dengan menggunakan metode *Levenshtein Distance*, satu-per-satu isi dari tabel KATA terfilter akan dibandingkan dengan kata kunci untuk memperoleh nilai jarak. Apabila dari setiap perbandingan yang dilakukan diperoleh nilai jarak berada pada kisaran 1-3 maka kata dari tabel KATA itu akan ditandai. Penetapan nilai jarak antara 1-3 dilandasi pemikiran bahwa apabila nilai jarak yang dihasilkan bernilai > 3 maka perbedaan antara kedua kata tersebut sudah terlalu besar. Selanjutnya dari semua kata yang tertandai hanya akan dipilih satu kata saja yang memiliki nilai jarak terkecil.

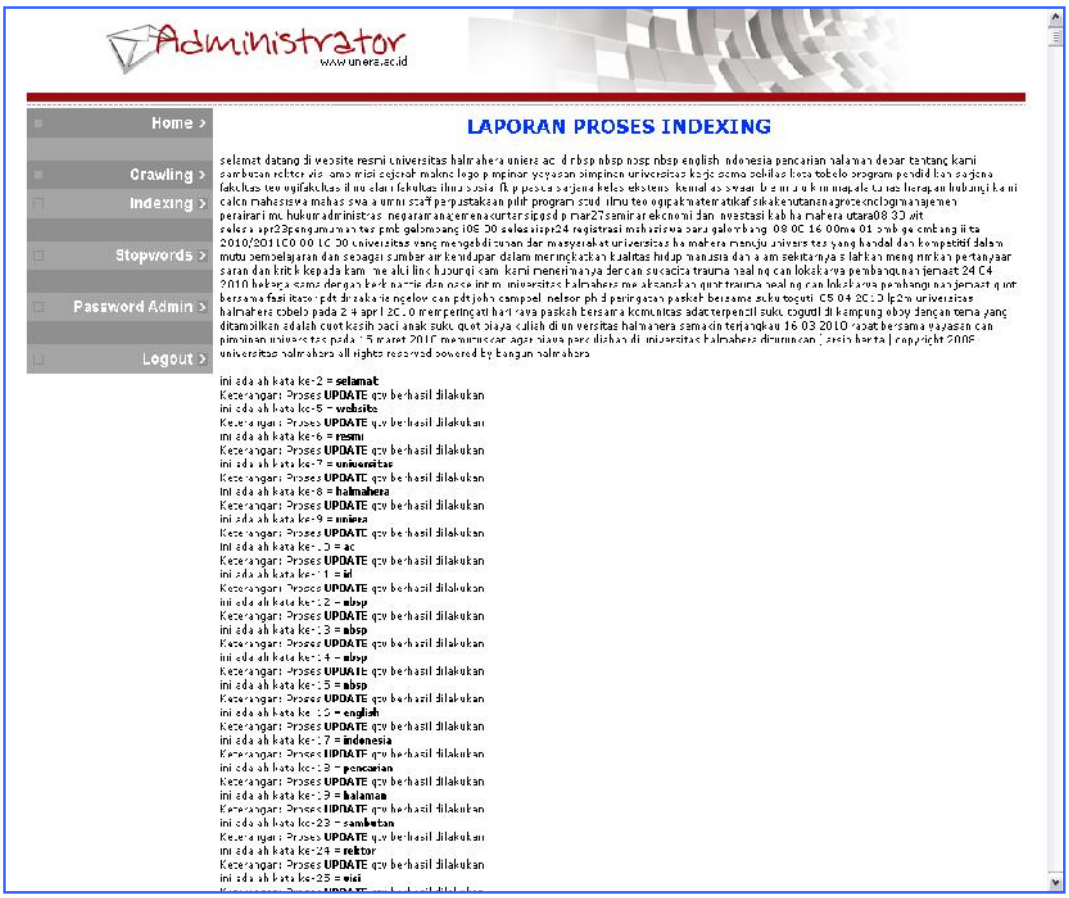
B. Hasil dan Pembahasan

Apabila proses *crawling* terhadap seluruh konten yang terdapat pada website selesai dilakukan maka admin akan mendapatkan laporan proses *crawling* seperti pada Gambar 6.1.

Gambar 6.1 Halaman hasil *crawling* administrator

Proses *indexing* akan memanfaatkan hasil yang diterima dari proses *crawling*. Pada proses ini seluruh kata yang terdapat pada website akan diindeks ke dalam tabel kata. Hasil dari

proses *indexing* dapat dilihat pada Gambar 6.3.



Gambar 6.3 Halaman hasil *indexing*

Proses pencarian dengan menggunakan kata kunci: **unverstas**, tidak mengembalikan hasil dikarenakan kata tersebut tidak mengenai target. Pada kondisi seperti ini sistem akan menganggap telah terjadi kesalahan dalam proses pengejaan kata kunci sehingga sistem memberikan saran kata kunci: **universitas**, dan 2 temuan teratas dari kata kunci tersebut seperti ditunjukkan pada Gambar 6.15 .

Gambar 6.15 Hasil pencarian dengan kata kunci berupa kata tidak berhasil

Pada kondisi ini sistem akan memberikan saran kata kunci baru yang merupakan pembetulan dari ejaan kata kunci sebelumnya. Saran kata kunci baru yang diberikan adalah: ***universitas*** yang merupakan pengkoreksian dari kata **unverstas**. Prosesnya dapat dijelaskan sebagai berikut:

Kata kunci = **unverstas**

– **Langkah 1**

Kata kunci = **unverstas**, dipecah ke dalam array sehingga:

Array ([0] => unverstas)

– **Langkah 2**

Untuk setiap isi array lakukan pencarian pada table CRAWL. Apabila pencarian bernilai *false* maka lakukan proses saranKata. Pada kondisi ini kata unverstas bernilai *false* sehingga proses saranKata akan dilakukan terhadap kata unverstas.

– **Langkah 3**

Filter semua kata pada tabel KATA yang memiliki panjang karakter (P) antara $P_{katakunci}$-3 sampai $P_{katakunci}$+3. Sehingga untuk kata unverstas dengan panjang 9 maka kata-kata yang terpilih adalah yang memiliki panjang antara 6 – 12.

– **Langkah 4**

Untuk setiap kata terfilter dari tabel KATA lakukan penghitungan jarak terhadap kata unverstas dengan metode *Levenshtein Distance*.

Misalkan tiga kata dari tabel kata yang akan dibandingkan adalah investasi (panjang kata = 9), universitas (panjang kata = 11) dan integritas (panjang kata = 10).

Tabel 6.1 Menghitung nilai jarak untuk kata investasi

		U	N	V	E	R	S	T	A	S
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
I	**1**	**1**	2	3	4	5	6	7	8	9
N	**2**	2	**1**	2	3	4	5	6	7	8
V	**3**	3	2	**1**	2	3	4	5	6	7
E	**4**	4	3	2	**1**	2	3	4	5	6
S	**5**	5	4	3	2	**2**	**2**	3	4	4
T	**6**	6	5	4	3	3	3	**2**	3	4
A	**7**	7	6	5	4	4	4	3	**2**	3
S	**8**	8	7	6	5	5	4	4	3	**2**
I	**9**	9	8	7	6	6	5	5	4	**3**

Tabel 6.2 Menghitung nilai jarak untuk kata universitas

		U	N	V	E	R	S	T	A	S
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
U	**1**	**0**	1	2	3	4	5	6	7	8
N	**2**	1	**0**	1	2	3	4	5	6	7
I	**3**	2	1	**1**	2	3	4	5	6	7
V	**4**	3	2	**1**	2	3	4	5	6	7
E	**5**	4	3	2	**1**	2	3	4	5	6
R	**6**	5	4	3	2	**1**	2	3	4	5
S	**7**	6	5	4	3	**2**	**1**	2	3	3

I	**8**	7	6	5	4	3	**2**	2	3	4
T	**9**	8	7	6	5	4	3	**2**	3	4
A	**10**	9	8	7	6	5	4	3	**2**	3
S	**11**	10	9	8	7	6	4	4	3	**2**

Tabel 6.3 Menghitung nilai jarak untuk kata integritas

		U	**N**	**V**	**E**	**R**	**S**	**T**	**A**	**S**
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
I	**1**	**1**	2	3	4	5	6	7	8	9
N	**2**	2	**1**	2	3	4	5	6	7	8
T	**3**	3	2	**2**	3	4	5	5	6	7
E	**4**	4	3	3	**2**	3	4	5	6	7
G	**5**	5	4	4	3	**3**	4	5	6	7
R	**6**	6	5	5	4	**3**	4	5	6	7
I	**7**	7	6	6	5	4	**4**	5	6	7
T	**8**	8	7	7	6	5	5	**4**	5	6
A	**9**	9	8	8	7	6	6	5	**4**	5
S	**10**	10	9	9	8	7	7	6	5	**4**

Dari perhitungan yang dilakukan melalui Tabel 6.1, Tabel 6.2 dan Tabel 6.3 di peroleh nilai jarak untuk setiap kata yang dibandingkan sebagai berikut:

LD(unverstas, investasi) = 3

LD(unverstas, universitas) = 2

LD(unverstas, integritas) = 4

– **Langkah 5**

Update nilai jarak dari semua kata di tabel KATA terfilter yang memiliki nilai jarak antara 1 – 3. Oleh karena sistem hanya menyimpan nilai jarak antara 1 – 3 maka hanya kata universitas dan investasi yang ter*update* nilai jaraknya pada table KATA (Gambar 6.17).

– **Langkah 6**

Urutkan isi tabel kata berdasarkan kriteria jarak terendah. Kata pada urutan teratas merupakan kata saran terpilih. Sehingga pada kasus ini kata ***universitas*** terpilih sebagai kata saran bagi kata **unverstas**.

C. Penutup

Kesimpulan

Dari hasil penelitian dan pengujian yang telah dilakukan dapat ditarik kesimpulan sebagai berikut:

1. Metode *Levenshtein Distance* dapat diterapkan untuk melakukan proses pengkoreksian kata kunci pada aplikasi pencarian berbasis web.
2. Memodifikasi perintah SQL: LIKE *condition* dari format LIKE '%katakunci%' menjadi LIKE '% katakunci %' (spasi diantara kata kunci dan *wildcard* %) akan mengembalikan hasil pencarian tanpa mengikutkan string yang memiliki substring yang memenuhi pola kata kunci.
3. Sistem dapat memproses inputan kata kunci dalam jumlah yang banyak dengan durasi waktu eksekusi yang tergantung dari kompleksitas kata kunci yang diinputkan.
4. Mengindeks frasa dengan status pencarian berhasil akan membantu dalam proses pengkoreksian frasa yang tidak memiliki spasi sebagai pemisah antar kata disamping juga akan mempersingkat waktu eksekusi.

Saran

Saran yang dapat diberikan dari hasil penelitian untuk pengembangan sistem ini ke depan adalah sebagai berikut:

1. Metode *Levenshtein Distance* mendukung operasi string sampai dengan 255 karakter sehingga dapat digunakan untuk operasi string pada skala yang lebih besar lagi.
2. Salah satu komponen penting dari sebuah mesin pencarian adalah bagaimana mengurutkan hasil pencarian (*ranking*). Sistem ini belum membahas penggunaan algoritma perankingan secara spesifik sehingga ke depannya perlu dikembangkan.
3. Mesin pencarian mengenal beberapa tanda yang biasanya digunakan untuk memaksimalkan hasil pencarian seperti penggunaan kata AND dan OR atau symbol (+) dan (–) yang belum diterdapat pada sistem ini.
4. Saat ini teknologi mesin pencarian sudah mampu melakukan pencarian *file* gambar yang dilakukan dengan memanfaatkan informasi nama *file* tersebut. Sistem ini belum memiliki kemampuan tersebut sehingga ke depannya perlu untuk dikembangkan.

Pustaka

Gilleland, M., **Levenshtein Distance in Three Flavors**, *http://www.merriampark.com/ld.htm*, diakses 5 Mei 2010.

Kaefer, H., 2003, **What Is a Web Crawler?,** *http://www.wisegeek.com/what-is-a-web-crawler.htm*, diakses 9 Mei 2010.

Nataliani Y., 2006, **Penerapan Algoritma Vektor untuk Mencocokkan String (String Matching) pada Finite Automata***, Tesis*, Program Studi S2 Ilmu Komputer, Universitas Gadjah Mada, Yogyakarta.

Navarro, G., Baeza-Yates, R., Sutinen, E., dan Tarhio, J., 2001, **Indexing Methods for Approximate String Matching**, Buletin IEEE Computer Society Technical Committee on Data Engineering.

Navarro, G., 2001, **A Guided Tour to Approximate String Matching**, ACM Comp. Surv., 33(1):31-88.

Schulz, K., dan Mihov, S., 2002, Fast **string correction with Levenshtein Automata**, International Journal on Document Analysis and Recognition, 5(1):67–85.

Sharma, S., 2008, **Web-Crawling Approaches in Search Engines**, Thesis, Computer Science and Engineering, Thapar University, Patiala.

Smith, S. E., 2003, **What is a Search Engine?,** http://www.wisegeek.com/what-is-a-search-engine.htm, diakses 8 Mei 2010.

Stephen, G. A.,1994, **String Searching Algorithms**, World Scientific Publishing Co. Pte. Ltd., Singapura.

Sulzberger, C., **The Levenshtein-Algorithm**, http://www.levenshtein.net, diakses 5 Mei 2010.

Ulman, C., 2007, **Beginning AJAX**, Wrox, ISBN: 0470106751.

Wang P, Berry M. W. dan Yang, Y., 2003, **Mining Longitudinal Web Queries: Trends and Patterns**, Journal of the American Society for Information Science and Technology, 54(8):743–758.