Analisis Algoritma Approximate String Matching Pada Fitur Autocorrect dalam Pencarian Data

Arie Satia Dharma[1*], Joko Banjarnahor[2], Okta Nainggolan[3], Yulianty Sihombing[4]

* Corresponding author: ariesatia@del.ac.id

Sejarah penerimaan

Diterima pertama kali: 15/09/2017

Diterima setelah perbaikan: 23/11/2017

Tanggal penerbitan: 05/04/2018

Abstract— *In searching, users will receive results that do not match what they want if they enter the wrong word on the search engine. There is an usefull applications for perform correction when searching a word that autocorrect feature. There are many string matching algorithms that can be used to apply the autocorrect feature. This paper applied a Levenshtein Distance algorithm and Hamming Distance algorithm on a prototipe of search engine. The performance of search engine is measured by using a comparison parameter that is running time, memory usage and accuracy. Keywords used in searching data is done by applying test scenarios on the scope of character substitution operation. Results obtained from this research is in terms of memory consumption and running time algorithm Hamming Distance is better than the Levenshtein Distance whereas in terms of accuracy Levenshtein algorithm is better than the Hamming Distance.*

Keywords— *Search Engine; Autocorrect; Levenshtein Distance Algorithm; Hamming Distance Algorithm*.

Intisari—Dalam melakukan pencarian, pengguna akan menerima hasil yang tidak sesuai dengan yang diinginkan jika memasukkan kata yang salah pada mesin pencari. Terdapat sebuah aplikasi yang berguna untuk melakukan pengkoreksian kata ketika melakukan pencarian yaitu fitur autocorrect. Ada banyak string matching yang dapat digunakan untuk menerapkan fitur autocorrect. Dalam penelitian ini digunakan algoritma Levenshtein Distance dan algoritma Hamming Distance pada prototipe mesin pencari. Kinerja mesin pencari diukur dengan menggunakan parameter pembanding yaitu running time, memory usage, dan accuracy. Kata kunci yang digunakan dalam pencarian data dilakukan dengan menerapkan skenario uji pada lingkup operasi penggantian karakter. Hasil yang diperoleh dari penelitian ini adalah dalam hal pemakaian memori dan running time Algoritma Hamming Distance lebih baik dari pada Levenshtein Distance sedangkan dalam hal akurasi algoritma Levenshtein lebih baik dari pada Hamming Distance.

Kata Kunci— Mesin Pencari; Autocorrect; Levenshtein Algorithm; Hamming Distance Algorithm.

I. PENDAHULUAN

Mesin pencari merupakan sebuah program komputer yang dapat digunakan untuk mengakses informasi yang dibutuhkan [9]. Dengan mengetikkan kata kunci yang diinginkan pada mesin pencari maka sistem akan menampilkan halaman yang berkaitan dengan kata kunci yang dimasukkan oleh pengguna. Namun, jika pengguna melakukan pencarian dengan kata kunci yang salah dalam pengetikan karakter maka hasil pencarian yang akan ditampilkan oleh mesin pencari tidak akan sesuai dengan yang dibutuhkan oleh pengguna [2]. Oleh karena itu, dibutuhkan sebuah fitur untuk menghasilkan pengkoreksian kata terhadap masukan yang diberikan oleh pengguna yaitu fitur autocorrect. Autocorrect digunakan oleh mesin pencari untuk membantu pengguna dalam pencarian suatu data [3]. Autocorrect adalah fitur untuk memeriksa kata kunci yang ingin dicari jika terdapat suatu kesalahan dalam pengejaan kata [4]. Pada penelitian ini, penulis akan melakukan analisa dan perbandingan terhadap Algoritma Levenshtein Distance dan Hamming Distance karena keduanya merupakan salah satu algoritma edit distance , dimana kedua algoritma tersebut bekerja dengan melakukan pencarian jumlah distance yang dimiliki oleh 2 buah string/objek. Kedua algoritma tersebut diterapkan pada fitur autocorrect dan hasil yang didapat dibandingkan untuk memperoleh accuration, running time dan memory usage yang digunakan oleh masing-masing algoritma..

II. TINJAUAN PUSTAKA

A. *Autocorrect*

Autocorrect merupakan fitur yang berguna untuk mengidentifikasi kata yang salah pengejaan dan membenarkan kata yang diketik pengguna secara otomatis [1]. Fitur ini akan memeriksa kata yang diketikkan oleh pengguna huruf demi huruf dan mencari kemiripan dengan huruf-huruf yang tersimpan pada database dimana data dengan kemiripan tertinggi akan ditampilkan sebagai layanan *autocorrect*.

[1, 2, 3, 4] *Program Studi Teknik Informatika Fakultas Teknik Informatika dan Elektro Institut Teknologi Del, Jln. Sisingamangaraja Sitoluama, Laguboti, Tobasa 22381 INDONESIA (+62 632 331234; e-mail: ariesatia@del.ac.id; if314020@students.del.ac.id; if314025@students.del.ac.id; if314003@students.del.ac.id)*

B. Algoritma *Levenshtein Distance*

Algoritma levenshtein distance bekerja menggunakan perhitungan jumlah perbedaan jarak antara dua string. Hasil dari perhitungan tersebut diperoleh dengan menentukan jumlah minimum dari operasi perubahan dari suatu string menjadi string yang lain. Terdapat 3 macam operasi yang dapat dilakukan oleh algoritma levenshtein distance yaitu operasi penghapusan karakter, operasi penggantian karakter, dan operasi penambahan karakter [6].

Dalam melakukan pencarian nilai minimal, algoritma levenshtein akan mengoptimalkan hasil pencarian sehingga algoritma ini tergolong program dinamis dalam pencarian nilai minimal [3]. Tabel 1 merupakan Pseudocode Algoritma Levensthein Distance.

TABEL I
PSEUDOCODE LEVENSTHEIN DISTANCE

```
int LevenshteinDistance(char s[1..m], char
t[1..n])
   // d is a table with m+1 rows and n+1
columns

   declare int d[0..m, 0..n]

   for i from 0 to m
       d[i, 0] := i
   for j from 0 to n
       d[0, j] := j

   for i from 1 to m
     for j from 1 to n
     {
       if s[i] = t[j] then cost := 0
                      else cost := 1
       d[i, j] := minimum(
       d[i-1, j] + 1,     // deletion
       d[i, j-1] + 1,     // insertion
       d[i-1, j-1] + cost // substitution
                          )
     }

   return d[m, n]
```

C. *Algoritma Hamming Distance*

Algoritma Hamming Distance mengukur distance string dengan mencari string yang sama dan mengkalkulasikannya terhadap jumlah string keseluruhan [23]. Jika nilai kedekatannya makin kecil,maka kedua string itu semakin dekat dan berlaku sebaliknya [24]. Operasi yang digunakan algoritma Hamming Distance hanyalah operasi penggantian karakter. Tabel 2 merupakan contoh ilustrasi Algoritma Hamming Distance.

TABEL III
ILUSTRASI ALGORITMA HAMMING DISTANCE

		1	2	3	4	5	6	7	8	9	10
T	=	V	O	L	K	S	W	A	G	E	N
S	=	V	O	L	K	X	W	E	G	E	N
						S		A			

D. Metode Akurasi

Metode pencarian akurasi yang di pakai untuk menentukan tingkat kedekatan antara nilai prediksi dengan nilai fakta didefinisikan sebagai berikut:

$$\text{Akurasi} = \frac{\text{Total Benar}}{\text{Total Benar} + \text{Total Tidak Benar}} \text{ x } 100\%$$

E. *Profiling Tools*

Profiling Tools merupakan *Testing Coverage Tools* untuk menilai suatu keefektifan suatu code [22]. *Profilling tools* biasanya digunakan untuk mendapatkan informasi terkait performace pada sebuah perangkat yang digunakan. Pada penelitian ini *profilling tools* digunakan sebagai *tools* yang berguna untuk melihat dan mengukur waktu tempuh (*running time*) yang diperlukan serta jumlah memori yang digunakan (memory usage) dalam menyelesaikan tugas pencarian..

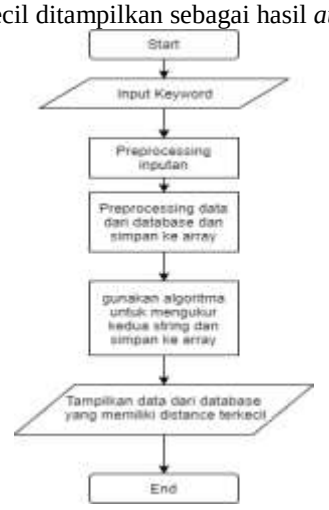
III. *ANALYSIS*

Paragraf harus teratur. Semua paragraf harus rata, yaitu sama-sama rata kiri dan dan rata kanan.

A. *Alur Kerja Autocorrect*

Pada fitur autocorrect diterapkan kedua algoritma pada saat aplikasi dijalankan. Berikut proses yang terjadi dalam fitur autocorrect :

1. Pengguna memberikan masukan berupa jenis tipe mobil pada kolom pencarian.
2. Setelah itu masukan tersebut di preprocessing.
3. Demikian juga data yang ada di dalam database, data tersebut juga akan melalui proses preprocessing dan disimpan didalam array.
4. Kemudian data masukan tersebut dibandingkan dengan semua tipe mobil yang terdapat di database menggunakan algoritma dalam aplikasi tersebut.
5. Hasil perbandingan di atas akan menghasilkan distance, dimana setiap hasil distance disimpan dalam array.
6. Setelah itu data dari database yang memiliki nilai distance paling kecil ditampilkan sebagai hasil *autocorrect*.



Gbr. 1 *Flowchart* dari *autocorrect*.

B. Analisis Data

Data yang digunakan sebagai data pencarian dalam penelitian ini adalah data kumpulan dari tipe mobil yang berjumlah sekitar 370.000 data, dengan kata kunci dari field column 'name'. Berikut gambaran dari data-data yang digunakan. Tabel 3 merupakan analisis data pada dataset dari setiap field dan tipe datanya:

TABEL IIIII
STRUKTUR DARI DATABASE MOBIL

No	Nama Kolom	Tipe data
1.	datacrawled	Datetime
2.	Name	Varchar
3.	Seller	Varchar
4.	offerType	Varchar
5.	Price	Int
6.	Abtest	Varchar
7.	vehicleType	Varchar
8.	yearOfRegistration	Int
9.	Gearbox	Varchar
10.	powerPS	Int
11.	Model	Varchr
12.	kilometer	Int
13.	monthOfRegistration	Int
14.	fuelType	Varchar
15.	Brand	Varchar
16.	notRepairedDamage	Varchar
17.	dateCreated	Datetime
18.	nrOfPictures	Int
19.	Postalcode	Int
20.	lastSeen	Varchar

C. *Pre-processing Process*

Pada tahap ini dilakukan proses penyederhanaan data menjadi bentuk yang lebih baku sehingga data lebih mudah untuk diproses, dengan cara melakukan penghapusan tanda baca, spasi (white space), pengubahan karakter huruf besar ke dalam huruf kecil setelah itu data disusun ke dalam array.

D. Pencarian *Distance* dari Kata Kunci

Pada tahap ini dilakukan proses pencarian distance menggunakan algoritma *Levenshtein Distance* dan *Hamming Distance*. Kata kunci yang dimasukkan oleh pengguna pada mesin pencari akan diolah dan dibandingkan satu persatu terhadap data yang terdapat pada database untuk mendapatkan nilai distance. Nilai distance terkecil yang diperoleh akan ditampilkan sebagai hasil dari *autocorrect*. Dalam melakukan perbandingan dari segi running time dan *memory usage* peneliti menggunakan 10 kata yang salah, dimana dilakukan skenario penggantian karakter. Dari kata ke-1 sampai kata ke-10 jumlah karakter yang diganti bertambah sebanyak 1 karakter. Jadi untuk kata yang ke-10 telah dilakukan penggantian karakter sebanyak 10 karakter. Sedangkan untuk perbandingan dari segi pengujian akurasi data yang digunakan adalah 360 jenis tipe mobil. Perhitungan tersebut akan dibagi ke dalam dua skenario yaitu skenario I dan skenario II. Skenario I dimana perbandingan akurasi menggunakan operasi kesalahan berupa penambahan karakter, pengurangan karakter, dan penggantian karakter. Sedangkan pengujian pada skenario II hanya menggunakan operasi penggantian karakter saja dengan jumlah data uji sebanyak 120 jenis tipe mobil.

E. Analisis Algoritma

Analisis dilakukan untuk mengetahui algoritma mana yang dapat menghasilkan kinerja lebih baik. Untuk mendapatkan kinerja tersebut maka dilakukan perbandingan dari hasil yang didapatkan pada pengujian. Parameter yang digunakan sebagai pembanding adalah *running time*, *memory usage*, dan *accuracy* dari rangkaian kerja *autocorrect* yang dijalankan oleh mesin pencari.

IV. IMPLEMENTASI

A. Implementasi Desain Antarmuka

Implementasi antarmuka dari penerapan Algoritma Levenshtein Distance dan Hamming Distance sebagai layanan autocorrect dapat dilihat pada Gambar 2 dan Gambar 3.

Gbr. 2 *Desain Antarmuka Penerapan Algoritma Levenshtein Distance*.

Gbr. 3 *Desain Antarmuka Penerapan Algoritma Hamming Distance*.

B. *Test Scenario Running Time* & Memori

Pada pengujian skenario ini, algoritma *Levensthein distance* dan algoritma *Hamming distance* akan dimplementasikan menjadi sebuah aplikasi *autocorrect* dan akan dilakukan *profilling* pada kedua algoritma tersebut menggunakan *tools profiling*. *Profiler* akan menghasilkan data waktu pencarian waktu dari setiap algoritma. Data skenario

pengujian *running time* dan memory dapat dilihat pada Tabel 4.

TABEL IVV
DATA SKENARIO PENGUJIAN RUNNING TIME & MEMORY

No	Input	Data Benar	Ganti Karakter
1.	Galf316	Golf_3_1.6	1
2.	RenauttCrio1.4	Renault_Clio_1.4	2
3.	Aude A1 Sportbakk Cline	Audi__A1_Sportback_S_line	3
4.	Smartkabrioblue/Cilber	Smart_Cabrio_Blau/Silber	4
5.	Mitsubissi Gallon AE0 v6	Mitsubishi_Galant_EA0_V6	5
6.	Aude G7 3 0 TDh PDF quartro_tiptronic	Audi_Q7_3.0_TDI_DPF_quattro_tiptronic	6
7.	Biate Fars ascudo und suche	Biete_Ford_Escort_und_suche	7
8.	Opal ostla q cappe 1.8 riter benziner	Opel_astra_g_coupe_1.8_liter_benziner	8
9.	BMW 520t tooling Pouluas stastang NOVA	BMW_530d_touring_Vollausstatung_NAVI	9
10.	Gulv 3 0 TDL Edision Naewartikel justand	Golf_2.0_TDI_Edition__Neuwertiger_Zustand	10

C. *Test Scenario Running Time* & Memori

Pada pengujian skenario ini akan dilakukan perbandingan akurasi dari 3 operasi yaitu: penambahan karakter, pengurangan karakter dan penggantian karakter. Kemudian juga akan dibandingan akurasi dengan hanya 1 komponen yaitu: penggantian karakter. Hal tersebut dilakukan agar dapat dilihat perbandingan persentase akurasi dari 3 komponen yang diuji dengan 1 komponen yang diuji. Pertimbangan untuk ini dikarenakan algoritma *Hamming Distance* hanya dapat melakukan penggantian karakter saja.

TABEL V
DATA SKENARIO PENGUJIAN AKURASI (PENAMBAHAN KARAKTER)

No	Data Pengujian	Data Asli	Tambah Karakter
1	audiaw624	audia624	1
2	bmw3323ci	bmw323ci	1
3	vwgol1f316	vwgolf316	1
4	hondaecrx16	hondacrx16	1
5	Smartcdaabrio	Smartcabrio	1
6	audiaa3	audia3	1
7	bmww316	bmw316	1
8	bmew328i	bmw328i	1
9	vwxgolf3	vwgolf3	1
10	bmw1120i	bmw120i	1

TABEL VI
DATA SKENARIO PENGUJIAN AKURASI (PENGURANGAN KARAKTER)

No	Data Pengujian	Data Asli	Kurang Karakter
1	bw116i	bmw116i	1
2	bmw52d	bmw520d	1
3	bmw50i	bmw520i	1
4	bw325i	bmw325i	1
5	ford12	fordka12	1
6	mada6	mazda6	1
7	bmw36i	bmw316i	1
8	Vwcraer	vwcrafer	1
9	bmw70iv8	bmw730iv8	1
10	hyundii30	hyundaii30	1

TABEL VII
DATA SKENARIO PENGUJIAN AKURASI (PENGGANTIAN KARAKTER)

No	Data Pengujian	Data Asli	Ganti Karakter
1	twiqgo16s	twingo16s	1
2	3ervwwolf	3ervwgolf	1
3	porsrhe911	porsche911	1
4	Opelsorsab	Opelcorsab	1
5	Suzukaslto	Suzukialto	1
6	aidia3	audia3	1
7	bmw116i	bmw316i	1
8	bnw520i	bmw520i	1
9	vwfolo10	vwpolo10	1
10	aidia618	audia618	1

D. *Test Scenario Akurasi II*

Pada pengujian skenario ini perbandingan penggantian karakter ditujukan untuk melihat perbandingan akurasi dari kedua algoritma sesuai operasi satu-satunya yang dimiliki algoritma Hamming Distance yaitu penggantian karakter. Jadi kedua algoritma di atas akan dibandingkan dari segi akurasi dengan hanya menggunakan operasi penggantian karakter. Untuk melihat hubungannya dengan Test Scenario Akurasi I sebelumnya maka pada pengujian ini data yang digunakan adalah sama dengan yang ada pada Tabel 7.

V. HASIL DAN PEMBAHASAN

Error! Reference source not found. merupakan hasil percobaan running time dari setiap algoritma setelah diubah kedalam bentuk grafik :

Gbr. 4 *Grafik Perbandingan Running Time*.

Pada **Error! Reference source not found.** Algoritma Levensthein memerlukan waktu yang lebih lama untuk

menghasilkan kata sebagai *autocorrect* dibandingkan dengan Algoritma Hamming Distance yang memerlukan waktu yang lebih sedikit.

Gbr. 5 *Grafik Perbandingan Memory Usage*.

Pada **Error! Reference source not found.** menunjukkan bahwa Algortima Levensthein lebih banyak menghabiskan memori dibandingkan dengan Algoritma Hamming Distance yang menggunakan memori yang lebih sedikit.

Dalam perbandingan dari segi akurasi I, digunakan data jenis tipe mobil yang dibagi 3 berdasarkan operasi pengujian akurasi I. Pada **Error! Reference source not found.**8 terdapat total hasil eksperimen pada akurasi dari komponen uji (penambahan karakter, penghapusan karakter, substitusi karakter) yang telah dilakukan:

TABEL VIII
HASIL PENGUJIAN AKURASI I (PENAMBAHAN, PENGURANGAN, DAN PENGGANTIAN)

No	Algoritma	Uji Akurasi	
		Benar	Salah
1	Levenshtein Distance	286	74
2	Hamming Distance	93	267

Pada **Error! Reference source not found.**9 terdapat hasil percobaan akurasi II dengan uji komponen penggantian karakter.

TABEL IX
HASIL PENGUJIAN AKURASI II (PENGGANTIAN KARAKTER)

No	Algoritma	Substitusi	
		Benar	Salah
1.	Levenshtein Distance	95	25
2.	Hamming Distance	93	27

Berdasarkan perbandingan *Running Time* diatas dapat dilihat bahwa rata rata waktu yang diperlukan Algoritma Levenshtein Distance untuk menghasilkan kata sebagai autocorrect lebih lama yaitu 1.201,8 ms dibandingkan waktu yang diperlukan Algoritma Hamming Distance yaitu 10,527 ms. Sementara Berdasarkan hasil perhitungan dapat dilihat bahwa rata rata memory yang diperlukan Algoritma Levenshtein Distance untuk menghasilkan kata sebagai autocorrect lebih banyak yaitu 6.553.652,2 byte dibandingkan memory yang diperlukan Algoritma Hamming Distance yang 3.479.294,2 byte.

Berdasarkan hasil jumlah data yang benar dan salah diatas dengan menggunakan perhitungan akurasi maka dapat diperoleh persentase keakuratan keseluruhan total data dari Algoritma Levenshtein Distance (79,4 %) dan Algoritma Hamming Distance (25,8 %). Persentase untuk skenario akurasi II adalah Algoritma Levenshtein Distance (79,1 %) dan Algoritma Hamming Distance (77,5 %).

Persentase akurasi untuk substitusi karakter menunjukkan bahwa Levensthein Distance dan Hamming Distance memiliki selisih akurasi sebesar 1,6%, dimana Levenshtein Distance memiliki tingkat akurasi lebih tinggi yaitu sebesar 79,1% dan Hamming Distance dengan tingkat akurasi sebesar 77,5%.

Sedangkan pada pada Tabel 10 persentase akurasi untuk total data diatas menunjukkan bahwa Algoritma Levenshtein Distance merupakan algoritma yang memiliki persentase paling tinggi dalam menghasilkan akurasi data yaitu 79,4% untuk hasil pencarian autocorrect daripada Algoritma Hamming Distance terutama yang memiliki persentase akurasi sebesar 25,8% terhadap perbaikan kata test case yang telah disediakan, hal itu dikarenakan algoritma Levenshtein unggul dalam setiap skenario test yang dilakukan yaitu penggantian karakter, pengurangan, dan penambahan.

VI. KESIMPULAN

Kesimpulan yang diperoleh dari hasil penelitian ini adalah:

1. Dalam pencarian running time dan memory diperoleh hasil dimana Hamming Distance memperoleh hasil yang lebih baik dari segi nilai running time maupun memory usage yang lebih sedikit yaitu 10,527 ms untuk running time dan 3.479.294,2 byte untuk memory usage dibandingkan Levenshtein Distance dengan 1.201,8 ms untuk running time dan 6.553.652,2 byte untuk memory usage.
2. Algoritma Levenshtein Distance memiliki keakuratan lebih tinggi yaitu 79,4% daripada algoritma Hamming Distance yang memiliki nilai akurasi 25,8% terhadap seluruh skenario test case yang disediakan.
3. Sedangkan pada uji coba akurasi dengan skenario penggantian karakter diperoleh akurasi Levenshtein Distance sebesar 79,1% dan Hamming Distance dengan tingkat akurasi sebesar 77,5%.

REFERENSI

[1] A. R. Yuliandaru, "Penerapan String Matching Pada Auto-Correct Berbasis Algoritma Levenshtein Distance," 2015.
[2] Benisius, "Sistem Pengkoreksian Kata Kunci dengan Menggunakan Metode Levenshtein Distance," Studi Kasus pada Website Universitas Halmamera.
[3] B. M. D. Adiwidya, "Algoritma Levensthein dalam Pendekatan Approximate String Matching," 2009.
[4] C. C. B, "Penerapan Algoritma Boyer Moore-Dynamic Programming untuk Layanan Auto-Complete dan Auto-Correct," Strategi Algoritma, 2011.
[5] Hajar, Teuku Ifghar, "Implementasi Algoritma Levenshtein Distance dan Boyer Moore untuk Fitur Autocomplete dan Autocorrect pada Aplikasi Katalog Perpustakaan Daerah Aceh Timur," 2015.
[6] I. W. S. A. M. Ni Made Muni Adriyani, "Implementasi Algoritma Levenshtein Distance dan Metode Empiris untuk Menampilkan Saran Perbaikan Kesalahan Pengetikan Dokumen Bahasa Indonesia," Universitas Udayana.

[7] L. Y.Banowosari, A. Darmawan dan K. Kurniawan, "Analisis pada Fitur Autocomplete Suggestion dan Semantik pada Pencarian di Mesin Pencari Google," vol. 8, p. 295, 2014.
[8] M. R. Pandiselvam.P, "A Comparative Study on String Matching Algorithms of Biological Sequences".
[9] M. Y. Soleh, "Implementasi Algoritma KMP dan Boyer-Moore dalam Aplikasi Search Engine Sederhana," 2011.
[10] N. B. P. Mahesh Lalwani, "Efficient Algorithm for Auto Correction Using n-gram Indexing," International Journal of Computer & Communication Technology, pp. 23-27
[11] R. K. Yeny Rochmawati, "Studi Perbandingan Algoritma Pencarian String dalam Metode Approximate String Matching untuk Identifikasi Kesalahan Pengetikan Teks," Jurnal Buana Informatika, 2016.
[12] R. Setiawan, "Penerapan Algoritma Boyer Moore pada Posting Twitter TMC Polda Metro Jaya untuk Melaporkan Kondisi Lalulintas dan Rute Jalan Kota Jakarta," Universitas Trilogi.
[13] Y. Primadani, "Simulasi Algoritma Levenshtein Distance untuk Fitur Autocomplete pada Aplikasi Katalog Perpustakaan," 2014.
[14] Z. A. A. Ardi Isbad Amar Gurning, "Penerapan Fuzzy String Matching pada Aplikasi Pencarian Tugas Akhir Mahasiswa Jurusan Sistem Informasi Berbasis Web (Studi Kasus: Fakultas Sains dan Teknologi UIN Suska Riau)".
[15] W. K. Dewanto and M. M. D. Widianta, "Pemanfaatan Fitur Proofing, Autocorrect, dan Auto Format pada Aplikasi Office dalam Meningkatkan Pelayanan Prima pada Lembaga Pelatihan Komputer," Jurnal Pengabdian Masyarakat J-DINAMIKA, vol. 1, p. 2, 2016.
[16] https://www.kaggle.com/orgesleka/used-cars-database, diakses pada tanggal 12 Januari 2017
[17] R. Ernawati, "Implementasi Algoritma Boyer Moore dan Metode N-GRAM untuk Aplikasi Autocomplete dan Autocorrect," Tugas Akhir, 2013.
[18] R. Setiawan, S. Setyaningsih dan A. Qur'ania, "Penerapan Algoritma Levenshtein Distance untuk Koreksi Ejaan Kata Berbahasa Indonesia," Jurnal Online Mahasiswa (JOM) Bidang Ilmu Komputer/Informatika, vol. 2, p. 2, 2014.
[19] H. Agung dan Y. Yogyawan, "Implementasi Boyer-Moore pada Aplikasi Pencarian Rumus Matematika dan Fisika," vol. 3, p. 1, 2016
[20] K. Alemerien dan K. Magel, "Examining the Effectiveness of Testing Coverage Tools: An Empirical",vol.8,p.5,2014.
[21] A. Budiman, "Implementasi Algoritma Hamming Distance dan Algoritma Brute Force Dalam Mendeteksi Kemiripan Source Code Bahasa C," Thesis Universitas Multimedia Nusantara., 2016.
[22] L. S. Putro, "Penerapan Kombinasi Algoritma Minhash Dan Binary Hamming Distance Pada Hybrid Perekomendasi Lagu," UNS-F. MIPA Jur. Informatika-M.0509043-2014, 2014
[23] A. Flaig, D. Hertl dan F. Krüger, "Evaluation von Java Profiler Werkzeugen," Abteilung Zuverlässige Softwaresysteme, 2013.
[24] https://docs.oracle.com/javase/8/docs/technotes/guides/visualvm/, diakses pada tanggal 7 Juni 2017.