Aplikasi Algoritma *Levenshtein* dalam Pencarian *Fuzzy* Artikel Blog

I Putu Bakta Hari Sudewa – 13521150[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
[1]13521150@std.stei.itb.ac.id

***Abstract*—Banyaknya informasi yang termuat dalam suatu laman web, seperti blog ataupun dokumentasi daring menjadikan fitur pencarian sebagai fitur wajib yang harus dimiliki laman tersebut. Algoritma *Levenshtein* merupakan salah satu algoritma yang dapat digunakan untuk membuat fitur pencarian menjadi lebih sangkil dan sesuai dengan ekspektasi pengguna. Selain itu, fitur pencarian *fuzzy* juga dapat memberikan pengalaman pengguna yang lebih baik. Pada suatu laman blog yang memiliki cukup banyak artikel, fitur pencarian dengan menggunakan algoritma *Levenshtein* digunakan untuk memperhitungkan kesalahan pengejaan dan variasi kata dalam *query* pencarian yang diberikan pengguna. Mulai dari judul, isi, penulis, hingga waktu dipublikasikannya suatu artikel dapat menjadi pilihan *query* pencarian yang dapat dilakukan pengguna.**

***Kata Kunci*—pencarian fuzzy, levenshtein, algoritma**

I. Pendahuluan

Informasi yang diperoleh melalui berbagai media daring merupakan makanan kita sehari-hari dewasa ini. Mulai dari bangun tidur di pagi hari hingga akan tidur kembali di malam hari, berbagai informasi baru tanpa henti terus diperbarui tiap detiknya.

Dahulu, jika ingin mendapatkan informasi terkait berita terbaru harus menunggu pagi di keesokan hari untuk dapat mengetahui informasi tersebut melalui surat kabar. Namun, sekarang hanya dengan membuka suatu situs berita daring, dapat dilihat berita terbaru terkait kejadian yang terjadi saat ini secara *realtime*.

Seiring dengan banyaknya informasi yang tersedia di dunia maya, seperti berita-berita, dokumentasi terkait teknologi baru yang diciptakan ataupun diperbarui, dan khususnya artikel-artikel blog yang dapat memberikan sudut padang baru bagi pembacanya, diperlukan suatu fitur yang perannya sangat penting bagi pengalaman pengguna yang ingin mencari tahu suatu informasi tertentu dari segudang informasi lain yang tersedia. Fitur pencarian merupakan fitur penting tersebut.

Fitur pencarian merupakan fitur penting yang wajib dimiliki oleh suatu situs yang memiliki segudang informasi, seperti situs blog, dokumentasi daring, dan sebagainya. Tentunya pencarian yang dapat dilakukan tidak sebatas hanya mencocokkan *query* pencarian dari pengguna huruf demi huruf, namun juga sebaiknya dapat memperhitungkan kesalahan pengejaan dan variasi kata dalam *query* tersebut. Pencarian yang dapat melakukan hal tersebut disebut dengan pencarian *fuzzy* atau pencarian abstrak/samar.

Pencarian *fuzzy* yang akan dibahas didasari oleh algoritma *Levensthein* yang dapat mengukur jarak kesamaan antara dua buah kata. Pendekatan yang dilakukan adalah menemukan berapa banyak koreksi yang harus dilakukan seperti menambahkan, menghapus, atau mengganti suatu huruf pada suatu kata agar kata tersebut sama dengan kata yang lain.

Selanjutnya, akan diperoleh persentase perbedaan antara *query* pencarian yang diberikan pengguna dengan, misalnya isi dari setiap artikel pada blog. Akan ditampilkan artikel yang memiliki hasil persentase kemiripan dengan batas tertentu, sehingga pengalaman pengguna yang dihasilkan dapat maksimal.

II. Landasan Teori

A. Program Dinamis

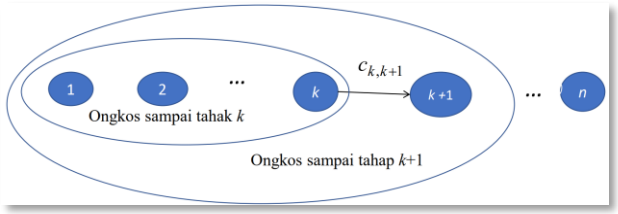
Program dinamis atau *dynamic programming* merupakan metode pemecahan masalah dengan cara menguraikan solusi menjadi sekumpulan tahapan (*stage*), sedemikian sehingga solusi persoalan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan.[1]

Kata "program" tidak ada kaitannya dengan pemrograman. Istilah "dinamis" muncul karena pencarian solusinya melakukan perhitungan dengan menggunakan tabel (yang dapat berkembang). Program dinamis digunakan untuk menyelesaikan persoalan-persoalan optimasi (maksimasi atau minimasi).[1]

Terdapat perbedaan antara algoritma *greedy* dengan program dinamis, yaitu pada algoritma *greedy* hanya satu rangkain keputusan yang dihasilkan sedangkan program dinamis lebih dari satu rangkaian keputusan yang dipertimbangkan.[1]

Pada program dinamis, rangkaian keputusan yang optimal dibuat dengan menggunakan prinsip optimalitas. Prinsip optimalitas, yaitu jika solusi total optimal, maka bagian solusi samptai tahap ke-k juga optimal. Prinsip optimalitas berarti bahwa jika kita bekerja dari tahap k ke tahap k + 1, kita dapat menggunakan hasil optimal dari tahap k tanpa harus kembali ke tahap awal.[1]

Ongkos pada tahap $k + 1 =$ (ongkos yang dihasilkan pada tahap k) + (ongkos dari tahap k ke tahap $k + 1$, atau $c_{k,k+1}$)



Gambar 1. Visualisasi penyeselaian masalah program dinamis tahap demi tahap
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf)

Terdapat beberapa karakterisitik persoalan dengan program dinamis, yaitu

1. Persoalan dapat dibagi menjadi beberapa tahap (*stage*), yang pada setiap tahap hanya diambil satu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status (*state*) yang berhubungan dengan tahap tersebut. Secara umum, status merupakan bermacam kemungkinan masukan yang ada pada suatu tahap.
3. Hasil dari keputusan yang diambil pada setiap tahap ditransformasikan dari status yang bersangkutan ke status berikutnya pada tahap berikutnya.
4. Ongkos (*cost*) pada suatu tahap meningkat secara teratur (*steadily*) dengan bertambahnya jumlah tahapan.
5. Ongkos pada suatu tahap bergantung pada ongkos tahap-tahap yang sudah berjalan dan ongkos dari tahap tersebut ke tahap berikutnya.
6. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahak k memberikan keputusan terbaik untuk setiap status pada tahap $k+1$.
7. Prinsip optimalitas berlaku pada persoalan tersebut.

Ada dua macam pendekatan dalam program dinamis, yaitu

1. Program dinamis maju (*forward* atau *up-down*), perhitungan dilakukan dari tahap $1, 2, \ldots, n-1, n$.
2. Program dinamis mundur (*backward* atau *bottom-up*), perhitungan dilakukan dari tahap $n, n-1, \ldots, 2, 1$.

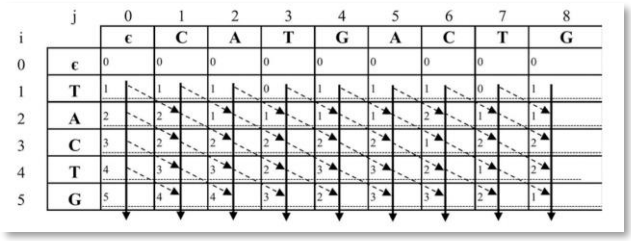
B. *Algoritma Levenshtein*

Algoritma *Levenshtein*, juga dikenal sebagai jarak edit *Levenshtein*, menghitung jumlah perbedaan antara dua kata yang disebut sebagai jarak. Variabel jarak ini juga digunakan sebagai parameter untuk mengecek sejauh mana perbedaan antara dua kata. Jarak *Levenshtein* antara dua kata adalah jumlah terkecil modifikasi karakter tunggal (penambahan, penghapusan, atau penggantian) yang diperlukan untuk mentransformasikan satu kata menjadi kata lainnya. Algoritma ini dinamai dari Vladimir Levenshtein, seorang matematikawan Soviet yang mempelajari algoritma ini pada tahun 1965.[2]

Secara umum ada dua tahap implementasi untuk menghitung jarak *Levenshtein* dari dua kata, yaitu membentuk matriks dengan membandingkan tiap huruf pada masing-masing kata dan memberikan nilai yang sesuai dari hasil perbandingan tersebut. Tahapan implementasi ini merupakan bagian dari program dinamis, karena prosesnya dibagi menjadi tahap-tahap dan setiap tahapnya akan diambil nilai minimum jarak *Levenshtein* untuk tahap tersebut.

Dalam bentuk matematis algoritma *Levensthein* menggunakan program dinamis direpresentasikan oleh formula berikut,

$$lev_{a,b}(i,j) = \begin{cases} lev_{a,b}(i-1,j-1), & a[j] = b[i] \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1 \end{cases}, & a[j] \neq b[i] \end{cases}$$



Gambar 2. Contoh matriks perhitungan jarak Levenshtein antara dua kata
(Sumber: Levenshtein Algorithm oleh Hrishitva Patel dan Goutham Ravichandran)

C. *Pencarian Fuzzy*

Pencarian *fuzzy* adalah metode pencarian yang memperhitungkan kesalahan pengejaan, variasi kata, atau perbedaan kecil dalam kata yang sedang dicari. Metode ini memungkinkan pencarian kata yang relevan meskipun terdapat kesalahan dalam *query* pencarian yang diberikan.

Implementasi dari pencarian *fuzzy* ini dapat dilakukan dengan menggunakan beberapa algoritma, seperti algoritma *Levenshtein*, algoritma *Soundex*, algoritma *Metaphone* dan *Double Metahphone*, dan *Cosine Similarity*. Umumnya pencarian *fuzzy* dilakukan jika menginginkan hasil pencarian yang fleksibel dan lebih akurat.[4]

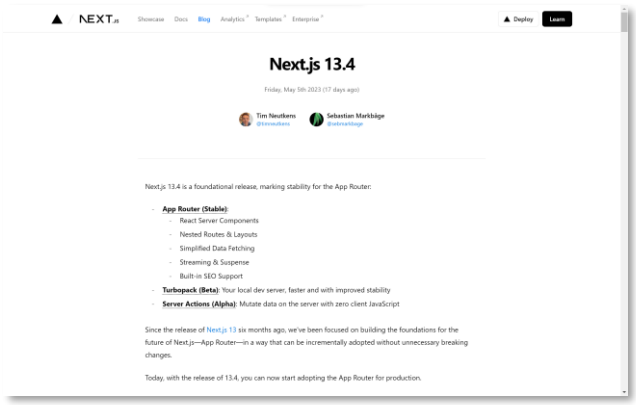
D. *Blog dan Artikel*

Istilah blog pertama kali digunakan oleh Jorn Barger pada tahun 1997. Jorn Barger menggunakan istilah ini untuk menyebut kelompok website pribadi yang selalu diperbarui secara terus menurus dan berisi link-link ke website lain yang dianggap menarik, disertai dengan komentar-komentar pemilik blog dan pengunjungnya.[3]

Blog kemudian berkembang mencari bentuk sesuai dengan keinginan para pembuatnya, yaitu para *blogger*. Blog yang pada mulanya merupakan catatan perjalanan seseorang di internet, yaitu tempat dikumpulkannya berbagai *link-link* yang mengarah ke situs-situs lain yang menarik.[3]

Dewasa ini blog bukan hanya sebatas buku harian daring yang dapad diakses orang-orang seperti saat pertama kali

dikenal. Namun, blog telah menjadi sarana untuk berbagi pengalaman dan memberikan tutorial-tutorial terkait suatu hal. Lebih daripada itu, blog juga telah digunakan sebagai sarana untuk *branding* dan mempublikasikan serta memaparkan hal baru yang telah dibuat oleh suatu perusahaan dan sebagainya.



Gambar 3. Contoh blog yang digunakan sebagai sarana publikasi versi baru suatu *framework*
(Sumber: dokumen pribadi penulis)

Untuk membuat suatu blog dapat menggunakan servis gratis yang disediakan oleh Google, yaitu Blogger, Wordpress, WiX, dan sebagainya. Namun, jika menggunakan servis-servis gratis tersebut, terdapat banyak batasan untuk membuat blog yang fleksibel yang sesuai keinginan.

Di balik layar suatu blog saat ini dapat berupa elemen statik berupa file markdown, html yang disimpan di server dan ditampilkan pada laman hingga elemen dinamis yang datanya disimpan pada suatu basis data dan diperbarui secara *realtime*.

Artikel pada blog adalah tulisan atau konten tertulis yang dipulikasikan di blog. Artikel biasanya berfokus kepada topik atau tema tertentu yang dirancang untuk memberikan informasi, pemikiran, atau pandangan tertentu kepada pembaca. Panjang dari suatu artikel dapat beragam, mulai dari beberapa paragraf hingga beberapa ribu kata tergantung pada kompleksitas topik yang dibahas. Tujuan utama artikel adalah memberikan nilai tambah kepada pembaca dengan memberikan informasi, pemikiran, atau pandangan yang relevan dan bermanfaat.

III. Analisis dan Pembahasan

A. *Implementasi Algoritma Levenshtein*

Pencarian *fuzzy* akan menggunakan algoritma *Levenshtein* sebagai algoritma utama untuk memperoleh persentase kemiripan hasil pencarian, yaitu *query* dari pengguna dan isi dari artikel pada blog.

Implementasi algoritma *Levenshtein* menggunakan pendekatan program dinamis. Dimulai dengan membuat suatu matriks berukuran $(n+1) \times (m+1)$, dengan n adalah jumlah huruf pada kata dalam *query* pencarian pengguna dan m adalah jumlah huruf pada kata dalam isi artikel blog. Untuk analisis kali ini, misalkan kata yang akan dibandingkan adalah "typescript" dengan "rust". "typescript" akan menjadi *query* pencarian pengguna dan "rust" adalah salah satu kata pada isi artikel blog.

Tahap pertama adalah melakukan inisialisai maktriks. "typescript" akan menjadi komponen kolom dan "rust" menjadi komponen barisnya. Terdapat representasi yang perlu diperhatikan saat melakukan pengisian matriks ini, yaitu sebagai berikut

penggantian	penghapusan
penambahan	(i, j)

Tabel 1. Representasi masing-masing sel di sekitar sel yang akan diisi saat ini, yaitu (i, j)

Sel ke-(i, j) pada matriks yang dibentuk menunjukkan jumlah operasi (penggantian, penghapusan, atau penambahan) yang diperlukan untuk mengubah sub-kata komponen baris menjadi sub-kata komponen kolom. Dalam analisis ini berarti, banyaknya operasi yang diperlukan untuk mengubah sub-kata dari "rust" menjadi sub-kata dari "typescript".

Tabel 1 menunjukkan representasi dari masing-masing sel di sekitar sel (i, j) saat ini, sel yang terletak pada indeks $(i-1, j-1)$ merepresentasikan *cost* atau banyaknya operasi penggantian, sel yang terletak pada indeks $(i-1, j)$ merepresentasikan banyaknya operasi penghapusan, dan sel yang terletak pada indeks $(i, j-1)$ merepresentasikan banyaknya operasi penambahan yang mungkin dilakukan.

Selanjutnya, akan dilakukan pengisian baris pertama dan kolom pertama matriks, sebagai berikut

		0	1	2	3	4
		" "	r	u	s	t
0	" "	0	1	2	3	4
1	t	1				
2	y	2				
3	p	3				
4	e	4				
5	s	5				
6	c	6				
7	r	7				
8	i	8				
9	p	9				
10	t	10				

Tabel 2. Pengisian baris dan kolom pertama matriks

Pengisian baris pertama matriks diperoleh dengan menghitung berapa banyak operasi yang diperlukan untuk mengubah sub-kata komponen baris hingga indeks ke *j* menjadi kata kosong (" "). Operasi yang sesuai untuk dilakukan adalah operasi penghapusan seperti representasi yang ditunjukkan pada **Tabel 1**.

Pada sel (0,0) banyaknya operasi yang diperlukan adalah 0 karena sub-kata komponen baris sama dengan sub-kata komponen kolom. Lalu pada sel (0,1) jumlah operasi yang diperlukan adalah 1 karena hanya diperlukan penghapusan satu buah huruf dari sub-kata komponen baris agar sama dengan sub-kata komponen kolom. Konsep yang sama berlaku untuk kolom barikutnya pada baris pertama.

Pengisian kolom pertama matriks memiliki konsep yang hampir serupa dengan pengisian baris pertama. Hanya terdapat perbedaan sudut pandang sehingga jenis operasi yang dilakukan berbeda, yaitu operasi penambahan. Pengisian kolom pertama menunjukkan berapa banyak huruf yang harus ditambahkan agar sub-kata (“ ”) pada komponen baris sama dengan sub-kata ke- i pada komponen kolom.

Untuk pengisian sel berikutnya, akan digunakan formula sebagai berikut,

$$lev_{a,b}(i,j) = \begin{cases} lev_{a,b}(i-1,j-1), & a[j] = b[i] \\ \min \begin{cases} lev_{a,b}(i-1,j)+1 \\ lev_{a,b}(i,j-1)+1 \\ lev_{a,b}(i-1,j-1)+1 \end{cases}, & a[j] \neq b[i] \end{cases}$$

dengan $lev_{a,b}(i,j)$ adalah banyaknya operasi hingga sel ke-(i,j). Dari persamaan di atas, terlihat jika huruf pada indeks ke-j dari komponen baris yang dalam hal ini dimisalkan sebagai a sama dengan huruf pada indeks ke-i dari komponen kolom yang dalam hal ini dimisalkan sebagai b, maka banyaknya operasi untuk sel ke-(i,j) sama dengan banyaknya operasi sel ke-$(i-1,j-1)$. Sedangkan, jika huruf pada indeks ke-j dari komponen baris yang dalam hal ini dimisalkan sebagai a tidak sama dengan huruf pada indeks ke-i dari komponen kolom yang dalam hal ini dimisalkan sebagai b, maka banyaknya operasi sel ke-(i,j) adalah nilai minimum dari operasi pada sel ke-$(i-1,j)+1$, $(i,j-1)+1$, dan $(i-1,j-1)+1$.

Maka, dengan menggunakan formula di atas, diperoleh matriks akhir sebagai berikut,

		0	1	2	3	4
		“ “	r	u	s	t
0	“ ”	0	1	2	3	4
1	t	1	1	2	3	3
2	y	2	2	2	3	4
3	p	3	3	3	3	4
4	e	4	4	4	4	4
5	s	5	5	5	4	5
6	c	6	6	6	5	5
7	r	7	6	7	6	6
8	i	8	7	7	7	7
9	p	9	8	8	8	8
10	t	10	9	9	9	8

Tabel 3. Matriks akhir yang terbentuk setelah proses pengisian

Dari **Tabel 3** diperoleh jumlah operasi minimum yang diperlukan untuk mengubah kata “rust” menjadi “typescript” adalah 8 operasi, ini ditunjukkan pada sel ke-(10,4).

Dengan mengetahui berapa banyak operasi minimum yang diperlukan untuk mengubah suatu kata menjadi kata lainnya, maka dapat diperoleh persentase kemiripan antara kedua kata tersebut, yaitu dengan membagi banyaknya operasi dengan jumlah huruf maksimum dari kedua kata atau secara matematis dirumuskan sebagai berikut,

$$\%\ kemiripan = \frac{banyaknya\ operasi}{\max(panjang\ kata1, panjang\ kata2)}$$

formula di atas juga disebut sebagai normalisasi jarak *Levenshtein*.

Berikut adalah kode implementasi algoritma *Levenshtein* untuk menghitung persentase kimiripan dua buah kata dalam bahasa pemrograman Typescript.

```
/**
 * Calculate match percentage between two string using
normalized levenshtein algorithm
 *
 * @param s_pattern Pattern string
 * @param s_target Target string
 * @returns Match percentage of pattern and target string
 */
export function levenshtein(s_pattern: string, s_target:
string): number {
  let operations_table: number[][] = new
Array(s_target.length + 1);

  for (let i = 0; i < operations_table.length; i++) {
    operations_table[i] = new Array(s_pattern.length +
1);
  }

  for (let i = 0; i < operations_table.length; i++) {
    operations_table[i][0] = i;
  }

  for (let i = 0; i < operations_table[0].length; i++) {
    operations_table[0][i] = i;
  }
  for (let row = 1; row < operations_table.length;
row++) {
    for (let col = 1; col < operations_table[0].length;
col++) {
      if (s_pattern[col - 1] === s_target[row - 1]) {
```

```
        operations_table[row][col] =
operations_table[row - 1][col - 1];
      } else {
        operations_table[row][col] =
          Math.min(
            operations_table[row - 1][col],
            operations_table[row][col - 1],
            operations_table[row - 1][col - 1]
          ) + 1;
      }
    }
  }

  return (
    (1 -
      operations_table[s_target.length][s_pattern.length] /
        Math.max(s_pattern.length, s_target.length)) *
    100
  );
}
```

Kode 1. Implementasi algoritma *Levenshtein* untuk menghitung persentase kemiripan dua buah kata dalam bahasa pemrograman Typescript

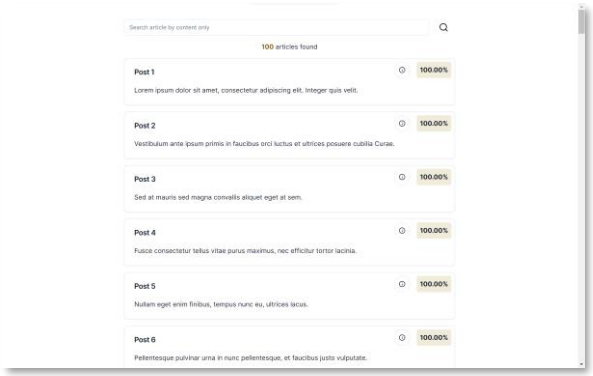
Setelah mengetahui bagaimana cara untuk memperoleh persentase kemiripan antara dua kata dengan algoritma *Levenshtein*, langkah berikutnya adalah menghubungkan hal tersebut dengan pencarian *fuzzy*.

B. Pembuatan Blog dan Fitur Pencarian Fuzzy

Selanjutnya akan dibuat sebuah blog untuk mengimplementasikan fitur pencarian *fuzzy* ini. Blog dibuat dengan menggunakan bahasa pemrograman Typescript dan dengan *framework* Next.js. Berikut adalah tampilan *landing page* dari blog tersebut, (blog dapat diakses di https://zuta-blog.vercel.app)

Gambar 4. Landing page blog
(Sumber: dokumen pribadi penulis)

Lalu berikut adalah tampilan halaman pencarian artikel,

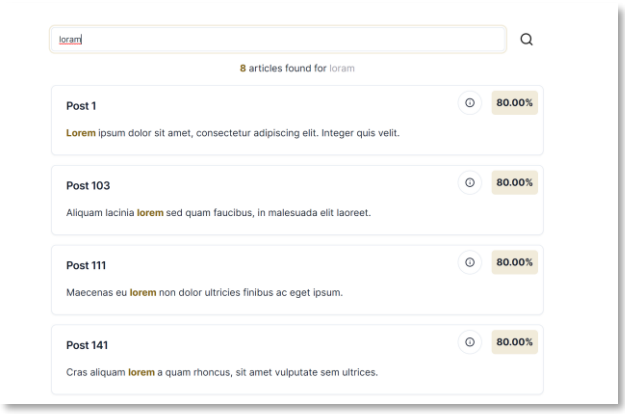


Gambar 5. Tampilan halaman pencarian artikel
(Sumber: dokumen pribadi penulis)

Blog tersebut telah diisikan artikel sebanyak 100 buah dengan panjang isi artikel sekitar 100 karakter. Pencarian *fuzzy* yang dapat dilakukan adalah pencarian terhadap isi dari artikel, tidak termasuk pencarian judul artikel.

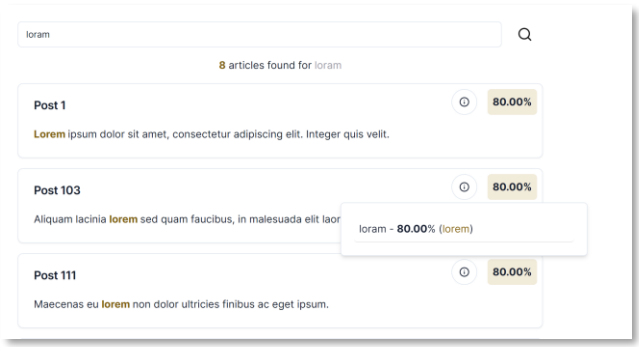
Algoritma *Levenshtein* pada bagian sebelumnya akan digunakan untuk memperoleh persentase kemiripan tiap kata pada isi artikel dengan tiap kata pada *query* pencarian pengguna. Misalnya pengguna melakukan pencarian “lorem”, maka akan dilakukan iterasi untuk setiap artikel yang ada. Lalu, isi dari setiap artikel tersebut selanjutnya dibentuk menjadi suatu array kata, jadi setiap kata yang dipisahkan oleh spasi akan dipisahkan dan dimasukkan ke dalam array tersebut. Begitu pula untuk *query* pencarian pengguna, tiap kata yang dipisahkan oleh spasi juga dibentuk menjadi suatu array. *Tracehold* atau batas minimum kemiripan tiap kata pada *query* pencarian pengguna dengan isi artikel adalah 60%. Batas ini digunakan dengan pertimbangan agar hasil pencarian tidak berlebihan dan masih tetap cukup fleksibel.

Setelah diperoleh persentase kimiripan setiap kata pada *query* pencarian dengan isi artikel maka artikel yang memiliki kata tersebut akan dikirimkan kembali menuju *client* untuk ditampilkan sebagai hasil pencarian. Berikut adalah contoh jika misalnya dilakukan pencarian kata “loram” pada blog.



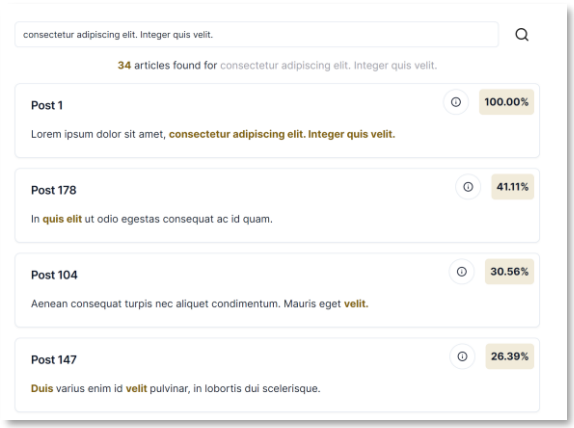
Gambar 6. Pencarian kata "loram" pada blog
(Sumber: dokumen pribadi penulis)

Dari **Gambar 6** terlihat ada 8 artikel yang ditemukan. Terlihat bahwa persentase kemiripan pencarian memiliki hasil maksimum 80% karena kata “loram” tidak ada pada seluruh isi artikel di blog, yang ada adalah “lorem” seperti ditunjukkan **Gambar 6**. Hal ini dapat dilihat secara lebih detail dengan menekan tombol i pada *card* artikel yang terletak di samping persentase kemiripan, ditunjukkan pada gambar di bawah,



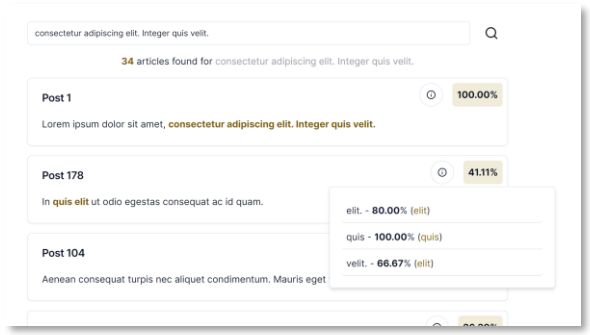
Gambar 7. Detail persentase kimiripan tiap kata
(Sumber: dokumen pribadi penulis)

Pada **Gambar 7** ditunjukkan detail persentase kimiripan kata “loram” dari *query* pencarian pengguna dengan kata “lorem” pada artikel dengan nilai sebesar 80%. Pencarian *fuzzy* pada blog ini tidak hanya sebatas pencarian satu kata saja, tapi juga dapat lebih dari satu kata, seperti contoh berikut,



Gambar 8. Pencarian dengan lebih dari satu kata
(Sumber: dokumen pribadi penulis)

Gambar 8 menunjukkan hasil pencarian untuk kalimat “consectetur adipiscing elit. Integer quis velit.”. Kalimat tersebut 100% sama dengan kalimat yang ada pada artikel dengan judul Post 1 sehingga diperoleh persentase kemiripan 100%. Namun, untuk artikel lainnya terlihat persentase kemiripan yang diperoleh jauh lebih rendah, yaitu 41,11% hingga 26,39%. Hal ini terjadi karena tidak seluruh kata pada *query* pencarian 100% sama dengan kata pada isi artikel.



Gambar 9. Detail pencarian kalimat "consectetur adipiscing elit. Integer quis velit."
(Sumber: dokumen pribadi penulis)

Gambar 9 menunjukkan tidak seluruh kata pada *query* pencarian memiliki persentase kemiripan di atas 60%, hanya ada tiga kata yang melewati batas minimum tersebut, yaitu “elit.”, “quis”, dan “velit”. Untuk kasus ini, persentase kemiripan yang ditampilkan merupakan jumlah persentase untuk tiap kata pada *query* pencarian yang berada di atas *tracehold* dibagi dengan banyaknya kata pada *query* pencarian. Dalam kasus ini, $(80 + 100 + 66.67)/6 = 44.11\%$.

IV. Kesimpulan

Pencarian *fuzzy* merupakan fitur yang wajib dimiliki oleh situs-situs web dengan segudang informasi berbeda, seperti dokumentasi daring ataupun blog. Dengan adanya fitur pencarian *fuzzy* ini, maka pengalaman pengguna yang dihasilkan akan maksimal. Algoritma *Levenshtein* sebagai fondasi dari fitur ini dapat menciptakan hasil pencarian yang fleksibel karena mampu memberikan persentase kemiripan antara *query* pencarian pengguna dengan data pada situs, bukan hanya sekadar mencocokkan huruf demi huruf. Selain itu, algoritma *Levenshtein* juga cukup mudah untuk diimplementasikan sehingga sangat cocok digunakan dalam projek skala kecil seperti pada blog pribadi.

V. Lampiran

1. Repositori github: https://github.com/sozyGithub/fuzzy-search-example
2. Blog: https://zuta-blog.vercel.app/blog
3. Youtube: https://www.youtube.com/watch?v=lHCBpakfu0I

Ucapan Terima Kasih

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa, karena atas bantuan dan rahmat-Nya lah makalah ini dapat diselesaikan. Penulis juga mengucapkan terima kasih kepada semua pihak yang telah menginspirasi dan membantu penulis salam proses penyusunan makalah ini. Terima kasih juga kepada Ibu Dr. Nur Ulfa Maulidevi, S.T, M.Sc., selaku dosen pengampu mata kuliah IF2211 Strategi Algoritma. Kepada para pembaca makalah ini, terima kasih telah meluangkan waktu untuk membaca, semoga informasi yang terkandung di dalam makalah ini bermanfaat.

DAFTAR PUSTAKA

[1] R. Munir, "Program Dinamis (Bagian 1)", *Informatika Rinaldi Munir*, 2023. [Online]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf [Diakses: 21 Mei 2023]

[2] H. Patel dan G. Ravichandran, "Levenshtein Algorithm", *Hcommons*. [Online]. Tersedia: https://hcommons.org/deposits/objects/hc:46164/datastreams/CONTENT/content [Diakses: 21 Mei 2023]

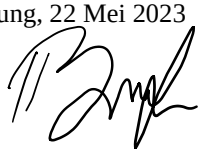
[3] Y. Panjaitan, "Mengenal Web Blog", *Leutikaprio*. [Online]. Tersedia: http://www.leutikaprio.com/main/media/sample/Mengelola%20Blog%20sebagai%20Media%20Pembelajaran%20Online%20SAMPLE%20DOWNLOAD.pdf [Diakses: 21 Mei 2023]

[4] V. P. Kuruvilla, "A Comprehensive Guide to Fuzzy Matching/Fuzzy Logic", *Nanonets*, 2022. [Online]. Tersedia: https://nanonets.com/blog/fuzzy-matching-fuzzy-logic/ [Diakses: 21 Mei 2023]

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, 22 Mei 2023



I Putu Bakta Hari Sudewa
13521150