Pengunaan Algoritma Wagner-Fischer untuk Melakukan Rekomendasi Kata

Fabian Savero Diaz Pranoto 13519140
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13519140@std.stei.itb.ac.id

***Abstract*—Pada zaman teknologi informasi ini, terdapat banyak teknologi yang menggunakan fitur *string matching* dan pengenalan kata untuk melakukan fungsi kerjanya. Contohnya adalah *search engine* atau mesin pencarian yang menerima masukan kata untuk melakukan pencarian dan juga terdapat *chatbot* yang mencari pola-pola pada masukan kata untuk menghasilkan sebuah keluaran. Akan tetapi, seringkali terdapat kesalahan penulisan kata atau *typo* yang dapat mencegah pelaksanaan fungsionalitas alat-alat tersebut secara optimal. Di makalah ini akan dibahas metode rekomendasi kata melalui pencarian *levenshtein distance* melalui penerapan Algoritma Wagner-Fischer yang tergolong algoritma pemrograman dinamis.**

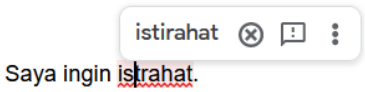
Keywords—string matching; levenshtein distance; Algoritma Wagner-Fischer; pemrograman dinamis;

I. PENDAHULUAN

Di zaman saat ini, informasi merupakan sebuah sumber daya yang sangat mudah dicari. Untuk mendapatkan informasi yang dibutuhkan dapat langsung menggunakan mesin pencarian yang banyak tersedia, contohnya mesin pencarian Google dan Yahoo. Dalam sehari, diperkirakan mesin pencarian Google memproses sekitar lima milyar *query* pencarian di seluruh dunia.

Dari sekian banyak jumlah pencarian yang dilakukan, tentu saja tidak semua kata kunci yang dimasukkan pada mesin pencarian memiliki ejaan yang benar. Apabila ejaan tidak benar, mesin pencarian dapat menampilkan hasil yang tidak dimaksud oleh pengguna. Untuk menanggulangi masalah ini, diterapkan fitur rekomendasi kata untuk menginformasikan pengguna apabila kata yang dimasukkan ada kemungkinan terjadi salah pengejaan.

Selain mesin pencarian kata, fitur ini juga dapat dilihat pada aplikasi Microsoft Word. Apabila kata yang dieja pengguna dinilai merupakan salah ejaan oleh aplikasi, akan diberikan rekomendasi kata kepada pengguna untuk memperbaiki kata yang diperkirakan salah ejaan.



Gambar 1: Contoh fitur rekomendasi kata

Untuk melakukan rekomendasi kata, dihitung jarak levenshtein antara dua kata yang mirip. Untuk mengimplementasikan ini akan digunakan Algoritma Wagner-Fischer yang merupakan algoritma golongan pemrograman dinamis.

II. TEORI DASAR

A. *Jarak Levenshtein*

Jarak Levenshtein merupakan sebuah angka yang menyatakan seberapa berbeda sebuah string a dengan sebuah string b. Jarak Levenshtein juga dikenal sebagai jarak minimum pengeditan sebuah kata. Pengeditan berupa penghapusan, insersi, dan substitusi sebuah karakter pada sebuah string atau kata. Semakin besar Jarak Levenshtein antara dua string, semakin berbeda kedua string tersebut. Fungsi untuk mencari Jarak Levenshtein antara sebuah string a dan sebuah string b sebagai berikut:

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

Gambar 2: Fungsi Jarak Levenshtein

Sumber:
https://en.wikipedia.org/wiki/Levenshtein_distance

Dari gambar di atas, variabel i dan j melambangkan posisi terminal karakter string a dan string b. Lambang $1(a_i=b_j)$ merupakan sebuah fungsi indikator yang bernilai 1 apabila a_i tidak sama dengan b_j dan bernilai nol apabila $a_i = b_j$.

Sebagai contoh, Jarak Levenshtein antara string “kitten” dan “sitting” diilustrasikan sebagai berikut:

1. kitten → sitten (subsitusi karakter “k” dengan “s”)
2. sitten → sittin (substitusi karakter “e” dengan “i”)
3. sittin → sitting (insersi karakter “g”)

Karena dibutuhkan 3 kali pengeditan, Jarak Levenshtein antara “kitten” dan “sitting” adalah 3. Untuk ilustrasi penyelesaiannya dalam bentuk matriks dapat melihat gambar di bawah.

		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

Gambar 3: Matriks Jarak Levenshtein

Sumber:

https://www.researchgate.net/figure/Examples-of-Levenshtein-Distance-calculation-between-strings_fig1_283041807

B. Pemrograman Dinamis

Pemrograman dinamis merupakan sebuah konsep algoritma yang ditemukan oleh seorang matematikawan bernama Richard Bellman pada 1950. Pemrograman dinamis menggunakan metode pemecahan masalah dengan cara menguraikan solusi permasalahan menjadi sekumpulan tahapan. Karena itu, solusi permasalahan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan. Program dinamis digunakan untuk menyelesaikan persoalan-persoalan optimasi.

Meskipun dari definisinya pemrograman dinamis terlihat mirip dengan algoritma golongan *greedy*, perbedaannya dengan algoritma *greedy* sebagai berikut:

1. Algoritma *Greedy* hanya menghasilkan satu rangkaian keputusan.
2. Pemrograman dinamis mempertimbangkan lebih dari satu rangkaian keputusan.

Proses penentuan rangkaian keputusan yang optimal pada pemrograman dinamis menggunakan Prinsip Optimalitas. Pengertian Prinsip Optimalitas sebagai berikut:

> Jika solusi total optimal, bagian solusi sampai tahap ke-k juga optimal.

Maksud dari ini adalah bahwa jika kita bekerja dari tahap ke-*k* ke tahap ke-*k*+1, kita dapat menggunakan hasil optimal dari tahap ke-*k* tanpa harus kembali ke tahap awal.

Karakteristik persoalan pada pemrograman dinamis sebagai berikut:

1. Persoalan dapat dibagi menjadi beberapa tahap, dan pada setiap tahap hanya diambil satu keputusan.
2. Masing-masing tahap terdiri dari sejumlah status 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 pada suatu tahap meningkat secara teratur dengan bertambahnya jumlah tahapan.
5. Ongkos pada sebuah tahap bergantung pada ongkos-ongkos tahap-tahap yang sudah berjalan.
6. Adanya hubungan rekursif yang mengidentifikasikan keputusan terbaik untuk setiap status pada tahap *k* memberikan keputusan terbaik untuk setiap status pada tahap *k*+1.

Pemrograman dinamis memiliki dua metode untuk menyelesaikan sebuah permasalahan. Kedua metode sebagai berikut:

1. Metode *top-down* atau maju menggunakan *memoization*. Pada metode ini, berusaha menyelesaikan sebuah masalah yang besar dengan mencari solusi terhadap sub-masalah yang lebih kecil. Metode untuk menyimpan solusi terhadap submasalah adalah *memoization*.
2. Metode *bottom-up* atau mundur menggunakan *tabulation.* Pada metode ini, merupakan kebalikan dari metode *top-down* dan biasanya menggunakan metode pengisian sebuah tabel berukuran *n* untuk menyimpan solusi yang dinamakan *tabulation*.

C. Algoritma Wagner-Fischer

Algoritma Wagner-Fisher merupakan sebuah algoritma golongan pemrograman dinamis yang dicetuskan oleh pasangan ilmuwan komputer bernama Robert A. Wagner dan Michael J. Fisher. Algoritma ini menghitung Jarak Levenshtein dua buah string menggunakan metode pemrograman dinamis. Kalkulasi jarak dilakukan melalui komputasi sebuah matriks

dua dimensi. Pseudocode Algoritma Wagner-Fischer sebagai berikut:

```
function WagnerFischer(a[1..m]:
character, b[1..n]: character) -> integer

  KAMUS
  d: array [0..m][0..n] of integer

  ALGORITMA
  // prefix sumber dapat diubah menjadi
string kosong dengan
  // menghapus semua karakter
  for i from 1 to m:
    d[i, 0] := i

  // prefix tujuan dapat dicapai dari
string kosong dengan
  // menginsersi semua karakter
  for j from 1 to n:
    d[0, j] := j

  // fungsi rekursi
  for j from 1 to n:
    for i from 1 to m:
      if a[i] = b[j]:
        substitutionCost := 0
      else:
        substitutionCost := 1

      d[i, j] := minimum(
         d[i-1, j] + 1,
         d[i, j-1] + 1,
         d[i-1, j-1] + substitutionCost)

  return d[m, n]
```

Dari pseudocode di atas, terlihat bahwa algoritma memiliki kompleksitas O(mn) karena mengiterasi melalui sebuah matriks 2D yang berukuran m x n.

D. *Fungsi Kemiripan*

Untuk melakukan rekomendasi kata, diperlukan sebuah fungsi yang akan memberi nilai matematis tingkat kemiripan antara dua buah string a dan b. Nilai kemiripan antara dua string harus memenuhi syarat sebagai berikut:

1. Dua string yang sangat berbeda akan memiliki nilai kemiripan 0.
2. Dua string yang sama akan memiliki nilai kemiripan 1.
3. Nilai kemiripan antar dua string memiliki nilai antara 0 sampai 1.

Untuk menentukan sebuah nilai persentase kemiripan antara dua string, dapat melakukan perbandingan Jarak Levenshtein kedua string dengan suatu nilai.

Ditentukan batas atas dan batas bawah Jarak Levesnhtein antara dua string. Batas bawah adalah 0 yang terjadi apabila kedua string mirip. Batas atas adalah nilai maksimum panjang kedua string. Oleh karena itu, dapat dibuat sebuah persamaan untuk menentukan nilai persentase kemiripan antara dua string.

$$s_{ab} = 1 - \frac{L_{ab}}{max(l_a, l_b)}$$

Gambar 4: Fungsi Nilai Kemiripan

Pada persamaan di atas, s_{ab} melambangkan nilai kemiripan string a dengan string b sedangkan L_{ab} adalah jarak Levenshtein. Lambang l_a dan l_b melambangkan panjang string a dan b yang akan dicari nilai maksimumnya untuk dibagi terhadap jarak Levenshtein.

III. Implementasi

A. *Langkah-Langkah Pemecahan Masalah*

Untuk menyelesaikan permasalahan perlu diformulasikan terlebih dahulu langkah-langkah untuk mendapatkan solusi. Pertama, pada penerapan dunia nyata program akan memiliki sebuah kamus yang berisi ejaan kata-kata yang benar. Untuk meminimasikannya penulis akan membuat kamus kata yang berisi 10 kata berupa kata kunci yang akan dipakai pada masukan program.

Kedua, dibuatkan sebuah struktur data yang menyimpan nilai kemiripan setiap kata pada kamus dengan string masukan pada program. Selain itu, perlu dibuatkan implementasi algoritma Wagner-Fischer dan implementasi fungsi yang akan menghitung nilai kemiripan antara kedua string.

Dalam pembuatan implementasi algoritma akan digunakan bahasa Java dan dibuat sebuah kelas bernama Rekomendasi yang akan memiliki metode sebagai berikut:

1. Metode untuk mencari jarak Levenshtein.
2. Metode untuk mencari nilai kemiripan.
3. Metode untuk menentukan rekomendasi kata terhadap kata salah ejaan.

B. *Penentuan Kamus*

Pada program ini, akan ditentukan 10 kata kunci yang akan digunakan sebagai kamus data pada program ini. Sepuluh kata tersebut adalah “deadline”, “diundur”, “dimajukan”, “selesai”, “minggu”, “hari”, “tambah”, “kapan”, “help”, dan “update”.

Implementasi kamus berupa sebuah array of string pada program sebagai berikut:

```
String[] kamus = {"deadline",
"diundur", "dimajukan",
"selesai", "minggu", "hari",
"tambah", "kapan", "help",
"update"};
```

C. *Implementasi Algoritma Wagner-Fischer*

Untuk penerapan Algoritma Wagner-Fischer, dibuatkan sebuah metode bernama LevDistance yang akan menerima dua buah parameter berupa string yang akan dibandingkan. Metode akan mengembalikan sebuah hasil berupa integer yang melambangkan jarak Levenshtein kedua string. Implementasi algoritma pada bahasa Java sebagai berikut:

```java
public int LevDistance(String str1, String str2) {
    int l1 = str1.length();
    int l2 = str2.length();

    if (l1 == 0)
        return l2;

    if (l2 == 0)
        return l1;

    int arr[][] = new int[l1 + 1][l2 + 1];

    for (int i = 0; i <= l1; i++)
        arr[i][0] = i;

    for (int j = 0; j <= l2; j++)
        arr[0][j] = j;

    for (int i = 1; i <= l1; i++) {
        char ch1 = str1.charAt(i - 1);

        for (int j = 1; j <= l2; j++) {
            char ch2 = str2.charAt(j - 1);

            int m = ch1 == ch2 ? 0 : 1;

            arr[i][j] = Math.min(
                Math.min((arr[i - 1][j] + 1),
                        (arr[i][j - 1] + 1)),
                        arr[i - 1][j - 1] + m);
        }
    }

    return arr[l1][l2];
}
```

Gambar 5: Fungsi Jarak Levenshtein

Pada cuplikan kode di atas, dihitung panjang setiap string. Apabila salah satu string memiliki panjang nol, jarak Levenshtein merupakan panjang string satunya. Apabila kedua string memiliki panjang yang tidak nol, dilakukan inisialisasi sebuah matriks berukuran *l1* x *l2*. Nilai awal pada baris pertama dan kolom pertama diinisialisasikan sebagai panjang prefix string. Setelah itu, dilakukan perhitungan setiap karakter pada string sesuai dengan fungsi yang telah dijelaskan pada Landasan Teori. Pada akhir perhitungan, jarak Levenshtein kedua string merupakan nilai pada baris *l*1 dan kolom *l*2 pada matriks.

D. *Implementasi Fungsi Nilai Kemiripan*

Untuk implementasi fungsi yang akan menghasilkan nilai kemiripan, dibuatkan sebuah metode bernama similarity yang akan menerima 3 parameter berupa kedua string yang dibandingkan beserta jarak Levenshtein-nya. Metode akan mengembalikan sebuah nilai bertipe double yang melambangkan nilai kemiripannya. Implementasi metode pada Bahasa Java sebagai berikut:

```java
public double similarity(String str1, String str2, int distance) {
    int l1 = str1.length();
    int l2 = str2.length();
    double ratio = (double) distance/Math.max(l1, l2);

    return 1 - ratio;
}
```

Gambar 6: Penerapan Fungsi Nilai Kemiripan

E. *Implementasi Fungsi Rekomendasi Kata*

Untuk implementasi fungsi rekomendasi kata, dibuat sebuah metode bernama recommend yang menerima satu parameter string masukan dari pengguna. Metode ini akan mengeluarkan output pada terminal berupa rekomendasi kata apabila tidak ada kata yang sama persis pada kamus. Apabila ada huruf pada kamus, akan mengeluarkan tulisan bahwa huruf ada pada kamus. Implementasi fungsi sebagai berikut:

```java
public void recommend(String str) {
    PriorityQueue<Entry> pq = new PriorityQueue<Entry>();

    for (String kata : kamus) {
        int distance = this.LevDistance(str, kata);
        double similarity = this.similarity(str, kata, distance);

        if (similarity > 0) {
            pq.add(new Entry(kata, similarity));
        }
    }

    try {
        Entry head = pq.poll();
        if (head.getValue() != 1) {
            System.out.println("\nApakah maksudmu " + head.getKey() +
"?");
        } else {
            System.out.println("\n" + head.getKey() + " ada di
kamus.");
        }

    } catch (Exception e) {
        return;
    }
}
```

Gambar 7 : Fungsi Rekomendasi Kata

Pada fungsi diatas, diinisialisasikan sebuah *priority queue* yang akan menerima elemen bertipe Entry. Tipe Entry ini

merupakan sebuah tipe yang memiliki atribut bertipe string yang melambangkan kata dan atribut bertipe double yang melambangkan nilai kemiripannya dengan string yang diuji. *Priority queue* akan mengurutkan elemen berdasarkan nilai kemiripan dari terbesar ke terkecil.

Untuk setiap kata pada kamus, akan dibandingkan terhadap string masukan dan dihitung nilai kemiripannya. Apabila nilai kemiripannya lebih dari nol, tambahkan pada *priority queue*. Setelah selesai mengecek setiap kata pada kamus, ambil elemen pertama pada *queue*. Apabila elemen pertama tidak null, cek apakah nilai kemiripannya 1 atau tidak. Apabila nilai kemiripannya 1, akan mengeluarkan pada terminal bahwa huruf ada pada kamus. Apabila nilai kemiripannya bukan 1, akan mengeluarkan rekomendasi kata yang merupakan elemen pertama tersebut.

```java
public class Entry implements Comparable<Entry> {
    private String key;
    private double value;

    public Entry(String key, double value) {
        this.key = key;
        this.value = value;
    }

    // getters
    public String getKey() {
        return this.key;
    }

    public double getValue() {
        return this.value;
    }

    @Override
    public int compareTo(Entry other) {
        if (this.value > other.value) return -1;
        if (this.value < other.value) return 1;
        else return 0;
    }
}
```

Gambar 8: Struktur Kelas Entry

F. *Program Utama*

Akan dilakukan penyatuan semua metode-metode pada kelas Rekomendasi pada program utama. Implementasi program utama sebagai berikut:

```java
public static void main(String[] args) {
    Rekomendasi rekomendasi = new Rekomendasi();
    Scanner scan = new Scanner(System.in);
    String input = scan.nextLine().toLowerCase();

    rekomendasi.recommend(input);
    scan.close();
}
```

Gambar 9: Program Utama

Pada program utama, pertama akan diinisialisasikan sebuah kelas Rekomendasi baru. Juga akan diinisialisasikan sebuah scanner untuk membaca masukan pengguna. Masukan pengguna akan diubah menjadi ke bentuk *lower case* dan akan dimasukkan pada pemanggilan metode recommend milik kelas Rekomendasi.

G. *Hasil Program*

Dilakukan pengujian terhadap program rekomendasi kata. Keluaran program akan memiliki 3 kemungkinan. Kemungkinan keluaran sebagai berikut:

1. Program berhasil melakukan rekomendasi kata.
2. Kata yang diuji berada pada kamus.
3. Tidak ada kata di kamus yang memiliki kemiripan sama sekali dengan kata yang diuji.

Pada pengujian pertama, dimasukkan sebuah masukan string “dedline” pada program. Hasil program utama sebagai berikut:

```
$ java Rekomendasi
dedline

Apakah maksudmu deadline?
```

Gambar 10: Pengujian Pertama

Terlihat bahwa berhasil melakukan rekomendasi untuk membenarkan kata “dedline” menjadi ejaan benar yaitu “deadline”. Isi *priority queue* pada pengujian pertama sebagai berikut:

deadline: 0.875
update: 0.2857142857142857
help: 0.2857142857142857
diundur: 0.1428571428571429
hari: 0.1428571428571429
dimajukan: 0.11111111111111116
kapan: 0.1428571428571429
selesai: 0.1428571428571429

Pada pengujian kedua, dimasukkan sebuah masukan string yang terdapat pada kamus yaitu string “diundur”. Hasil program sebagai berikut:

```
$ java Rekomendasi
diundur

diundur ada di kamus.
```

Gambar 11: Pengujian Kedua

Karena string “diundur” sudah ada pada kamus, program akan mengatakan bahwa string tersebut sudah sesuai dengan yang ada di kamus. Isi *priority queue* pada pengujian ini sebagai berikut:

diundur: 1.0
minggu: 0.2857142857142857
dimajukan: 0.33333333333333337
deadline: 0.125
update: 0.1428571428571429

Pada pengujian ketiga, dimasukkan sebuah masukan string "zzzzzzz" pada program. Hasil program utama sebagai berikut:

Gambar 12: Pengujian Ketiga

Pada pengujian ini, tidak mengeluarkan keluaran sama sekali karena tidak ada kata yang mirip pada kamus sehingga isi *priority queue* kosong. Karena *priority queue* kosong, program akan langsung keluar.

IV. Kesimpulan dan Saran

Algoritma Wagner-Fischer dapat digunakan untuk melakukan rekomendasi kata pada berbagai keadaan dan aplikasi. Meskipun begitu, terdapat kemungkinan optimasi yang dapat dilakukan terhadap algoritma Wagner-Fischer. Algoritma memiliki kompleksitas waktu O(mn) dengan m dan n melambangkan panjang kedua string yang dicek. Melalui pengamatan yang lebih dalam, ditemukan bahwa algoritma hanya membutuhkan baris saat ini dan sebelumnya yang disimpan pada suatu saat. Oleh karena itu, kompleksitas waktu algoritma dapat dikurangi menjadi O(n).

Keakuratan rekomendasi kata sudah lumayan bagus namun dapat diperbagus dengan memberi batas bawah sebuah nilai kemiripan agar dapat meningkatkan akurasi yang direkomendasikan. Selain memberikan sebuah *constraint* berupa batas bawah untuk meningkatkan akurasi rekomendasi, dapat mengubah algoritma untuk melakukan *approximate string matching* yaitu sebuah teknik untuk mencari string yang cocok terhadap suatu pola secara kira-kira. Algoritma yang dipakai penulis saat ini mencocokkan string terhadap suatu string lain secara eksak. Dengan menerapkan *approximate string searching* program dapat mengenali kata-kata yang merupakan sebuah *prefix* atau sebuah *suffix* dari string di kamus.

Video Link at Youtube

https://youtu.be/c5yS62zK0DY

Ucapan Terima Kasih

Puji syukur penulis haturkan kepada Tuhan Yang Maha Esa karena tanpa karunia-Nya, hampir tidak mungkin penulis menyelesaikan laporan penelitian yang berjudul Penggunaan Algoritma Wagner-Fischer untuk Melakukan Rekomendasi Kata. Laporan penelitian ini merupakan salah satu tugas mata kuliah Strategi Algoritma pada Semester Genap Tahun Akademik 2020/2021 di Institut Teknologi Bandung.

Terselesaikannya penulisan karya tulis ini juga tidak terlepas dari bantuan maupun arahan beberapa pihak. Oleh karena itu, penulis menyampaikan terima kasih kepada:

1. Orang tua yang selalu memberikan doa serta dukungan kepada penulis dalam menyelesaikan karya tulis ilmiah ini.
2. Bapak Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc., Ph.D. selaku dosen mata kuliah Strategi Algoritma yang telah membimbing dan mengajari penulis selama satu semester sehingga penulis dapat memahami semua materi perkuliahan yang diajarkan.
3. Bapak Dr. Ir. Rinaldi Munir, MT. selaku pengurus website yang berisi slide materi kuliah yang sangat membantu penulis di pembelajaran semester ini serta dalam pembuatan karya tulis ini.
4. Teman-teman saya yang telah membantu dan memberi pendapat terhadap karya tulis penulis selama tahap penulisan.

References

[1] Siregar, T.D. Kesumo, *Levenshtein Distance Calculation Using Dynamic Programming for Source Code Plagiarism Checking*, Teknik Informatika ITB, 2010.

[2] Gardahadi, *Plagiarism Detection Using Levenshtein Distance With Dynamic Programming*, Teknik Informatika ITB, 2019.

[3] Munir, Rinaldi, *Diktat Kuliah IF2211 Strategi Algoritma*, Teknik Informatika ITB, 2021.

[4] Stanford CS124 Lecturer, *Minimum Edit Distance*, diakses dari https://www.stanford.edu/ pada 10 Mei 2021.

[5] GeeksForGeeks, *Java Program to Implement Wagner and Fisher Algorithm for Online String Matching*, diakses dari https://www.geeksforgeeks.org/java-program-to-implement-wagner-and-fisher-algorithm-for-online-string-matching/ pada 10 Mei 2021.

[6] Nam, Ethan, *Understanding the Levenshtein Distance Equation for Beginners*, diakses dari https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0 pada 10 Mei 2021.

[7] Educative, *Grokking Dynamic Programming Patterns for Coding Interviews*, diakses dari https://www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2G1pAq0OO0 pada 10 Mei 2021.

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, 11 Mei 2021

Fabian Savero Diaz Pranoto 13519140