Pengaplikasian Fitur Koreksi Otomatis pada Pengolah Kata dengan Memanfaatkan Algoritma KMP

Sharon Bernadetha Marbun - 13519092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519092@std.stei.itb.ac.id

***Abstrak*—Pengolah kata merupakan aplikasi yang digunakan untuk mempermudah proses pengolahan kata seperti penyusunan, penyuntingan, pemformatan, dan pencetakan teks. Tidak jarang aplikasi ini dilengkapi dengan fitur-fitur atau perangkat lunak bahkan perangkat keras yang mendukung pengefektifan pengolahan teks seperti koreksi otomatis, pengenalan suara, *planning* dan *outlining*¸ serta pemberian umpan balik. Koreksi otomatis merupakan fitur yang digunakan untuk memperbaiki kesalahan pengetikan atau pengejaan yang umum dilakukan oleh pengguna. Koreksi otomatis seperti yang terdapat pada perangkat lunak *Microsoft Office Word* dilakukan dengan menyelaraskan hasil pengetikan pengguna dengan entri yang disimpan pada daftar koreksi otomatis yang apabila dideteksi akan digantikan dengan perbaikan kata yang juga berdasarkan data yang disimpan pada daftar koreksi otomatis. Fitur koreksi otomatis ini dapat diterapkan dengan memanfaatkan algoritma pencocokan *string*, salah satunya algoritma *Knuth-Morris-Pratt* (KMP).**

Kata kunci — pengolah kata; koreksi otomatis; pencocokan string; algoritma Knuth-Morris-Pratt (KMP)

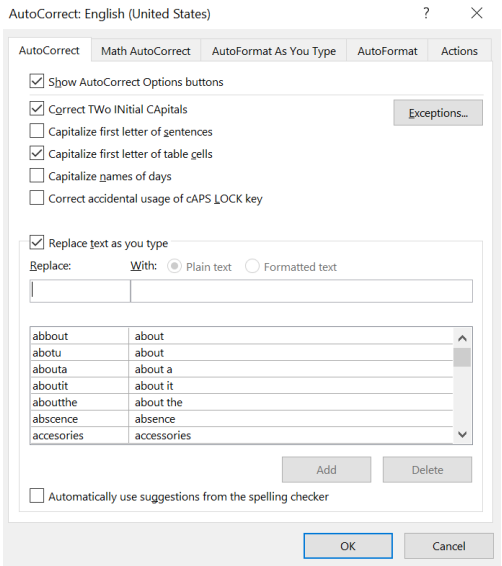
I. PENDAHULUAN

Seiring berkembangnya teknologi, kebiasaan menulis di atas kertas mulai digantikan oleh kebiasaan mengetik secara digital di atas *keyboard*. Tidak hanya dalam memenuhi kewajiban tertentu, kebiasaan ini juga dilakukan dalam mendukung aktivitas sehari-hari, seperti misalnya pembuatan pengingat di *smartphone*, membuat catatan kelas untuk koleksi pribadi di laptop, mengirimkan pesan kepada teman lewat *chat*, dan lain sebagainya.

Faktor yang mendukung terjadinya perubahan ini salah satunya adalah kenyamanan pengguna dan kemudahan dalam pengolahan teks yang didukung oleh aplikasi pengolah kata. Pengolah kata atau yang dalam bahasa Inggris disebut dengan istilah *word processor* adalah perangkat lunak yang dibangun untuk mendukung pengguna dalam melakukan pekerjaan yang meliputi penyusunan, penyuntingan, pemformatan, dan pencetakan teks. Aplikasi ini dapat memudahkan pengguna karena dilengkapi dengan fitur-fitur yang mengefektifkan pengolahan teks serta biasanya dikemas bersama-sama dengan perangkat lunak atau perangkat keras lain yang mendukung (Morphy & Graham, 2011: 642). Beberapa fitur tersebut antara lain pengecekan pengejaan dan tata bahasa atau koreksi otomatis, pengenalan suara, *planning* dan *outlining*, serta pemberian umpan balik pada bagian tertentu dari teks yang sedang dikerjakan. Beberapa aplikasi pengolah kata yang dapat ditemui adalah *Microsoft Word*¸ *Microsoft 365*¸ *Pages*, *Docs To Go*¸ *Notepad*¸ *Google Docs*, *Ez Word*¸*Open Office Writer*, *Atlantis Word Processor*, dan *WPS Writer*.

Koreksi otomatis merupakan fitur yang digunakan untuk memperbaiki kesalahan pengetikan oleh pengguna secara otomatis yang biasa ditemukan pada aplikasi pengolah kata dan antarmuka pengeditan teks untuk *smartphone* dan juga komputer. Tujuan utamanya adalah untuk memudahkan pengguna dalam menghindari kesalahan pengetikan yang umum dilakukan. *Se*cara *default*, koreksi otomatis pada dilakukan berdasarkan daftar koreksi otomatis yang disediakan oleh sistem. Terkadang, disediakan juga pilihan bagi pengguna untuk menghapus atau menambahkan entri ke dalam daftar koreksi otomatis sesuai keinginan.

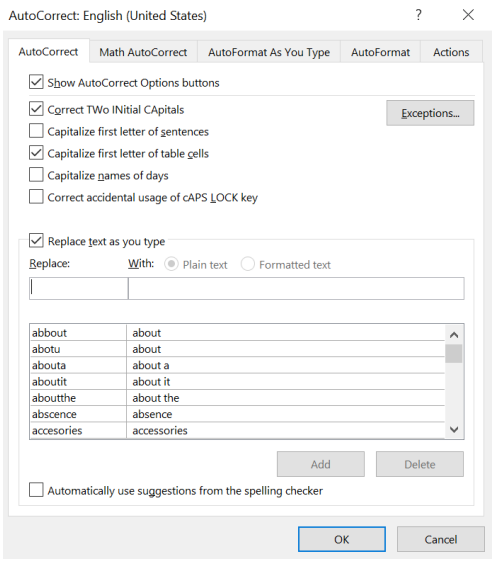
Salah satu aplikasi pengolah kata yang menyediakan fitur koreksi otomatis adalah *Microsoft Office Word*. Fitur ini dapat diatur melalui menu *File* → *Options* → *Proofing* → *AutoCorrect Options.*



Gambar 1.1 *AutoCorrect* pada *Microsoft Office Word*

Sumber: dokumentasi penulis

Selain untuk perbaikan kesalahan pengetikan, koreksi otomatis juga dapat dimanfaatkan sebagai jalan pintas untuk menuliskan simbol-simbol matematika ataupun simbol lainnya yang tidak terdapat pada papan *keyboard*, seperti yang disediakan pada fitur *Math AutoCorrect* oleh *Microsoft Office Word* sebagai berikut.



Gambar 1.2 *Math AutoCorrect* pada *Microsoft Office Word*

Sumber: dokumentasi penulis

Fitur koreksi otomatis ini dapat diterapkan dengan memanfaatkan algoritma pencocokan *string* sehingga masukan pengguna yang salah dapat langsung digantikan dengan *string* perbaikan. Oleh karena itu, pada makalah ini akan dibahas mengenai pengaplikasian fitur koreksi otomatis pada pengolah kata dengan memanfaatkan salah satu algoritma pencocokan *string* yaitu algoritma *Knuth-Morris Pratt* (KMP).

II. LANDASAN TEORI

A. *Pencocokan String*

Berdasarkan *Dictionary of Algorithms and Data Structures, National Institute of Standards and Technology (NIST)*, pencocokan *string* (*string matching*) merupakan sebuah permasalahan menemukan kemunculan string *pattern* (pola) di dalam string lain.

Pencocokan *string* secara garis besar dibedakan menjadi dua yaitu sebagai berikut (Syaroni & Munir, 2005: 1).

1. *Exact string matching*, merupakan pencocokan *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama.
2. *Inexact string matching* atau *fuzzy string matching*, merupakan pencocokan *string* secara samar, seperti sebagai berikut.
    a. Pencocokan *string* berdasarkan kemiripan penulisan (*appeoximate string matching*), merupakan pencocokan *string* dengan dasar kemiripan dari segi kemiripan (jumlah karakter, susunan karakter dalam dokumen).
    b. Pencocokan *string* berdasarkan kemiripan ucapan (*phonetic string matching*), merupakan pencocokan *string* dengan dasar kemiripan dari segi pengucapannya.

Beberapa algoritma yang dapat digunakan untuk melakukan pencocokan *string* antara lain *brute force*, *Knuth-Morris-Pratt*, *Boyer-Moore*, *Zhu-Takaoka*, *quick search*, pencarian string menggunakan *deterministic finite automata*, *Karp-Rabin*, *Shift-Or*, *Aho-Corasick*, algoritma *Smith*, dan *strsrch*.

B. *Algoritma Knuth-Morris Pratt (KMP)*

Algoritma KMP merupakan algoritma pencocokan string yang mengubah dikembangkan oleh Donaled E. Knuth pada tahun 1967 dan James H. Morris pada tahun 1966 secara terpisah, yang dipublikasikan secara bersamaan pada tahun 1977.

Pada algoritma ini, konsep *prefix* dan *suffix* digunakan untuk menentukan berapa banyak pergeseran yang perlu dilakukan. Pergeseran dilakukan sebanyak mungkin sedemikian sehingga tidak diperlukan pengulangan pemeriksaan yang mubazir. Misalnya dilakukan pencocokan string antara *string pattern* P dengan *string* teks T. Jika terjadi ketidakcocokan pada T dan P pada T[i] dan P[j] (T elemen ke-i dan P elemen ke-j), maka jumlah pergeseran yang perlu dilakukan adalah sebanyak panjang *prefix* P[0..j-1] terbesar yang juga merupakan *suffix* dari P[1..j-1].

Untuk mempermudah pencarian panjang *prefix* terbesar yang juga merupakan *suffix*, digunakan konsep fungsi pinggiran atau *border function* b(k). Fungsi pinggiran b(k) adalah ukuran terbesar pada P[0..k] yang juga merupakan *suffix* dari P[1..k].

Keterangan:

j = posisi terjadinya ketidakcocokan pada *pattern*

k = posisi sebelum terjadinya ketidakcocokan pada *pattern*

Berikut merupakan contoh tabel yang memuat nilai fungsi pinggiran untuk *pattern* “abaaba”.

(k = j-1)

j	0	1	2	3	4	5
P[j]	a	b	a	a	b	a
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

b(k) is the size of the largest border.

Gambar 2.1 Tabel nilai fungsi pinggiran untuk *string* “abaaba”

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Langkah-langkah yang diperlukan dalam menemukan nilai fungsi pinggiran seperti pada gambar 2.1 adalah sebagai berikut.

1. j = 1 → k = 0

   rentang *prefix* : a

   rentang *suffix* : -

   irisan *prefix* dan *suffix* terpanjang = -

   b(k) = 0

2. j = 2 → k = 1

   rentang *prefix* : ab

   rentang *suffix* : b

   irisan *prefix* dan *suffix* terpanjang = -

   b(k) = 0

3. j = 3 → k = 2

   rentang *prefix* : aba

   rentang *suffix* : ba

   irisan *prefix* dan *suffix* terpanjang = a

   b(k) = 1

4. j = 4 → k = 3

   rentang *prefix* : abaa

   rentang *suffix* : baa

   irisan *prefix* dan *suffix* terpanjang = a

   b(k) = 1

5. j = 5 → k = 4

   rentang *prefix* : abaab

   rentang *suffix* : baab

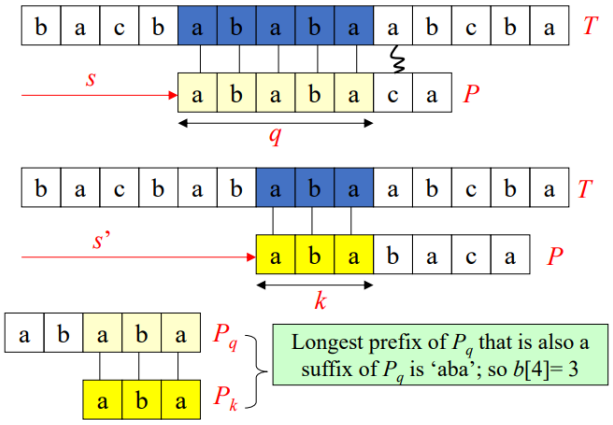
   irisan *prefix* dan *suffix* terpanjang = ab

   b(k) = 2

Langkah-langkah pencocokan *string* dengan algoritma KMP adalah sebagai berikut.

1. Ditentukan *string pattern* dan teks yang hendak diperiksa.
2. Dilakukan perhitungan fungsi pinggiran untuk setiap elemen dari *string pattern*.
3. Dilakukan pencocokan *string* mulai dari karakter *pattern* dan teks yang paling kiri
4. Apabila terjadi ketidakcocokan, pemeriksaan selanjutnya dilakukan setelah *pattern* digeser sebanyak nilai dari fungsi pinggiran b(k) dengan k = j-1 dan j merupakan posisi dimana dilakukan pemeriksaan terhadap *pattern*. Dengan kata lain, *pattern* digeser sebanyak nilai dari fungsi pinggiran dari elemen *pattern* terakhir sebelum terjadinya *mismatch*.
5. Pemeriksaan terus dilakukan hingga *pattern* ditemukan pada teks atau pemeriksaan sudah sampai ke elemen terakhir teks.

Berikut adalah contoh penerapan algoritma KMP dalam penyelesaian pencocokan *pattern* P “ababaca” dengan teks T “bacbababaabcba”.



Gambar 2.2 Penyelesaian pencocokan *pattern* “ababaca” dengan teks “bacbababaabcba” menggunakan algoritma KMP

Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Pada gambar di atas, dapat kita lihat bahwa terdapat ketidakcocokan antara elemen ke-5 P dengan elemen ke-9 T (perhitungan indeks dimulai dari 0). Oleh karena itu, *pattern* digeser ke kanan sebesar b(5-1) = b(4), yaitu sebesar 3 (rentang *prefix* = ababa, rentang *suffix* = baba, dengan irisan antara *prefix* dan *suffix* terpanjang = aba).

Setelah dilakukan pergeseran, maka dilakukan lagi pencocokan *string* T dan P satu per satu mulai dari kiri hingga ditemukan *pattern* “ababaca” pada teks atau pemeriksaan sudah dilakukan pada elemen terakhir teks.

III. IMPLEMENTASI PROTOTIPE KOREKSI OTOMATIS

Prototipe dari koreksi otomatis dapat diimplementasikan secara sederhana dengan membuat sebuah program yang menerima masukan dari pengguna, memeriksanya berdasarkan *string pattern* yang tersedia (berupa kesalahan umum yang dilakukan oleh pengguna dan disimpan pada basis data) yang apabila ditemukan akan digantikan dengan *corrected string* (juga disimpan pada basis data yang bersesuaian dengan *pattern*). Setelah masukan pengguna diperiksa dan diperbaiki, hasil perbaikan kemudian ditampilkan ke layar sebagai keluaran.

A. *Implementasi Algoritma KMP*

Fungsi KMP berikut diimplementasikan dalam bahasa pemrograman Python yang mengembalikan *boolean True* apabila *string pattern* ditemukan di dalam *text*, dan mengembalikan *False* apabila tidak.

```
def KMPMatch(pattern, text):
```

```
    #mengembalikan indeks dari karakter pertama
string matching antara pattern dengan text
    n = len(text)
    m = len(pattern)
    fail = getLPS(pattern)
    i=0
    j=0
    while i < n:
        if (text[i] == pattern[j]):
            if (j == m-1):
                return True
            i+=1
            j+=1
        elif j > 0:
            j = fail[j-1]
        else:
            i+=1
    return False
```

Berikut merupakan beberapa fungsi pelengkap yang digunakan untuk mengimplementasikan algoritma KMP yang juga ditulis dalam bahasa pemrograman Python.

1) Fungsi yang mengembalikan array LPS (elemennya berisi nilai fungsi pinggiran untuk setiap karakter pada string pattern)

```
def getLPS(pattern):
    #mengembalikan array yang memuat jumlah
irisan terbesar antara prefix dan suffix
    length = len(pattern)
    lps = [0 for _ in range(length)]
    for i in range(1,length):
        curr = getPref(pattern,i+1)
        max = 0
        for j in range(i):
            if compPrefSuf(curr,j):
                max = j
        lps[i] = max
    return lps
```

2) Fungsi yang mengembalikan prefix dari sebuah kata

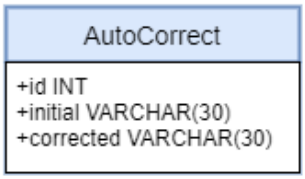
```
def getPref(word, len):
    #mengembalikan prefix sepanjang len dari
sebuah string word
    pref = ''
    for i in range(len):
        pref += word[i]
     return pref
```

3) Fungsi yang memeriksa apakah prefix dan suffix dari sebuah string bernilai sama untuk panjang tertentu

```
def compPrefSuf(word, length):
    #mengembalikan true apabila prefix dan
sufix sepanjang length sama
    pref = ''
    suf = ''
    for i in range(length):
        pref += word[i]
    for i in range(len(word)-
length,len(word)):
        suf += word[i]
    return (pref == suf)
```

B. Basis Data

Basis data digunakan untuk menyimpan daftar koreksi otomatis yang digunakan sebagai acuan dalam penyelarasan koreksi yang dilakukan. Skema basis data yang digunakan adalah sebagai berikut.



Gambar 3.2.1. Skema basis data

Sumber: dokumentasi penulis

- id merupakan integer yang digunakan sebagai penanda unik untuk setiap entri daftar koreksi otomatis
- initial merupakan *string pattern* yang akan dicocokkan ke masukan pengguna yang merupakan kesalahan pengetikan/pengejaan
- corrected merupakan *string* yang merupakan hasil perbaikan dari *string* initial apabila ditemukan pada masukan pengguna

Data pada tabel AutoCorrect merujuk pada daftar koreksi otomatis yang digunakan dalam pengolah kata *Microsoft Office Word (English: United States).*

id	initial	corrected
1	abbout	about
2	abotu	about
3	abouta	about a
4	aboutit	about it
5	aboutthe	about the
6	abscence	absence
7	accesories	accessories
8	accidant	accident
9	accomodate	accommodate
10	accordingto	according to
11	accross	across
12	acheive	achieve
13	acheived	achieved
14	acheiving	achieving
15	acn	can
...	...	...
894	you;d	you'd
895	you;re	you're
896	youare	you are
897	your a	you're a
898	your an	you're an
899	your her	you're her
900	your here	you're here
901	your his	you're his
902	your the	you're the
903	your their	you're their
904	your your	you're your
905	youve	you've
906	ytou	you
907	yuo	you
908	yuor	your

Gambar 3.2.2. Isi tabel AutoCorrect

Sumber: dokumentasi penulis

C. *Main Program*

Algoritma dari *main program* adalah sebagai berikut.

```
while True:
    masukan = input("Input: ")
    for i in range(len(initial)):
        if KMPMatch(initial[i], masukan):
            masukan = masukan.replace(initial[i],corrected[i])
    print("Output: " + masukan)
    print()
```

Gambar 3.3.1 Algoritma *main program*

Dalam sebuah *infnite loop*, pengguna diminta untuk memasukkan *input* berupa teks yang akan dikoreksi. Setelah itu, dilakukan pemeriksaan *string pattern* terhadap masukan pengguna. *Pattern* yang digunakan adalah setiap elemen dari kolom initial. Oleh karena itu, dilakukan pencocokan *string* menggunakan algoritma KMP untuk setiap elemen kolom initial dengan masukan pengguna di dalam *for-loop*. Apabila *pattern* ditemukan pada masukan pengguna, maka *pattern* tersebut akan digantikan dengan elemen dari kolom corrected yang memiliki id yang sama dengan elemen kolom initial yang bersesuaian.

D. *Hasil Pengujian*

Hasil dari pengujian program yang dilakukan adalah sebagai berikut.

1) *Hasil pengujian 1*

```
Input: I"m very suprised to know that yuor favorite caharcter isthe one that I hate the most.
Output: I'm very surprised to know that your favorite character is the one that I hate the most.
```

Gambar 3.4.1 Hasil pengujian 1

Sumber: dokumentasi penulis

2) *Hasil pengujian 2*

```
Input: How abbout goign tot he developors' office to search for difefrent documnets?
Output: How about going to the developers' office to search for different documents?
```

Gambar 3.4.2 Hasil pengujian 2

Sumber: dokumentasi penulis

3) *Hasil pengujian 3*

```
Input: It won;t be wasted to give such much efort since yoiu'll still get an unforgettable experienc anyway.
Output: It won't be wasted to give such much effort since you'll still get an unforgettable experience anyway.
```

Gambar 3.4.3 Hasil pengujian 3

Sumber: dokumentasi penulis

4) *Hasil pengujian 4*

```
Input: He doens't deserve the award he wnated, who;s with me?
Output: He doesn't deserve the award he wanted, who's with me?
```

Gambar 3.4.4 Hasil pengujian 4

Sumber: dokumentasi penulis

5) *Hasil pengujian 5*

```
Input: I spent hte wohle yera wokring at home.
Output: I spent the whole year working at home.
```

Gambar 3.4.5 Hasil pengujian 5

Sumber: dokumentasi penulis

6) *Hasil pengujian 6*

```
Input: What are you doing?
Output: What are you doing?
```

Gambar 3.4.6 Hasil pengujian 6

Sumber: dokumentasi penulis

7) *Hasil pengujian 7*

```
Input: What are you douing?
Output: What are you douing?
```

Gambar 3.4.7 Hasil pengujian 7

Sumber: dokumentasi penulis

8) Hasil pengujian 8

```
Input: Saya suka teh manis ini.
Output: Saya suka the manis ini.
```

Gambar 3.4.8 Hasil pengujian 8
Sumber: dokumentasi penulis

E. Pembahasan

Perbaikan kesalahan pengetikan dilakukan apabila masukan pengguna mengandung *string pattern* yaitu *string* yang disimpan pada kolom initial pada tabel AutoCorrect. Apabila *pattern* ditemukan di dalam masukan pengguna setelah dilakukan pencocokan *string* dengan algoritma KMP, *pattern* tersebut kemudian digantikan dengan *string* perbaikan yaitu *string* yang terdapat pada kolom corrected dengan id yang sama.

Analisis dari hasil pengujian adalah sebagai berikut.

1. Pada hasil pengujian 1 sampai 5, terlihat bahwa koreksi otomatis menjalankan fungsinya dengan baik.
2. Pada hasil pengujian 6, tidak terdapat kesalahan penulisan sehingga *input* sama dengan *output.*
3. Pada hasil pengujian 7, terdapat kesalahan pengetikan yang dilakukan oleh pengguna, tetapi karena *string* "douing" tidak terdaftar pada daftar koreksi otomatis tepatnya pada kolom initial, maka perbaikan tidak dilakukan.
4. Pada hasil pengujian 8, tidak terdapat kesalahan penulisan namun dilakukan koreksi otomatis karena *pattern* "teh" yang disimpan pada kolom initial pada basis data terdeteksi pada masukan pengguna. Ini merupakan kasus dimana koreksi otomatis dilakukan tidak sesuai dengan keinginan pengguna.

IV. KESIMPULAN

Algoritma pencocokan *string* seperti algoritma KMP dapat diterapkan dalam pengaplikasian fitur koreksi otomatis karena fitur ini melakukan *exact matching* terhadap masukan pengguna. Apabila pada masukan pengguna ditemukan *pattern* yang terdaftar pada daftar koreksi otomatis, *pattern* tersebut kemudian digantikan dengan *string* perbaikan yang juga disimpan pada daftar koreksi otomatis untuk *pattern* yang sesuai.

Program koreksi otomatis hanya mengandalkan daftar koreksi otomatis yang tersedia pada sistem dan tidak menerapkan konsep *machine learning*. Maka, agar fitur koreksi otomatis dapat menjalankan fungsinya dengan baik, pembuat program harus dapat dengan akurat mengetahui kesalahan-kesalahan pengetikan atau pengejaan yang umum dilakukan dan memasukkannya secara manual ke dalam daftar koreksi otomatis.

Dapat disimpulkan bahwa program ini tidak dapat secara sempurna memperbaiki kesalahan pengetikan dan bisa saja justru menyulitkan pengguna ketika pengguna sebenarnya hendak mengetik *string* tertentu yang terdaftar sebagai "kesalahan pengetikan" oleh sistem. Supaya fitur koreksi otomatis dapat memberikan keuntungan, saran yang diberikan oleh penulis adalah pengguna diharapkan dapat menyesuaikan daftar koreksi otomatis dengan kebiasaan kesalahan pengetikan yang dimiliki, dengan menghapus ataupun menambahkan entri dari daftar koreksi otomatis sesuai kebutuhan.

V. UCAPAN TERIMA KASIH

Pertama-tama, penulis mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa karena atas hikmat dan berkat yang diberikan-Nya, penulis dapat menyusun makalah ini hingga selesai. Penulis juga mengucapkan terima kasih yang sebesar-besarnya kepada para dosen pengampu mata kuliah IF2211 Strategi Algoritma atas ilmu dan bimbingan yang diberikan yang menjadi modal utama bagi penulis dalam menyelesaikan makalah ini. Tak lupa juga, penulis mengucapkan terima kasih kepada orang tua, saudara, dan rekan-rekan yang senantiasa mendukung dan memberikan motivasi kepada penulis.

LINK VIDEO YOUTUBE

https://youtu.be/-zqjUqxPCZU

REFERENSI

[1] Dictionary of Algorithms and Data Structures. National Institute of Standards and Technology. https://xlinux.nist.gov/dads/. Diakses tanggal 10 Mei 2021 pukul 20.00 WIB.

[2] Morphy, P. dan Steve Graham. 2011. *Word processing programs and weaker writers/readers: a meta-analysis of research findings*. Springer Science+Business Media.

[3] Munir, R. 2021. *Bahan Kuliah IF2211 Strategi Algoritma: Pencocokan String (String/Pattern Matching)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. Diakses tanggal 10 Mei 2021 pukul 21.00 WIB.

[4] Qwords.2021.*10 Aplikasi Pengolah Kata Terbaik untuk Profesional*. https://qwords.com/blog/aplikasi-pengolah-kata/. Diakses tanggal 11 Mei 2021 pukul 08.00 WIB.

[5] Syaroni, Mokhamad dan Rinaldi Munir. 2005. *Pencocokan String berdasarkan Kemiripan Ucapan (Phonetic String Matching) dalam Bahasa Inggris.* Yogyakarta: Seminar Nasional Aplikasi Teknologi Informasi 2005 (SNATI 2005).

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.

Balige, 11 Mei 2021

Sharon Bernadetha Marbun
13519092