Penerapan Algoritma String Matching untuk Deteksi Kesalahan Penulisan dan Identifikasi Kata Koreksi

Shazya Audrea Taufik - 13522063
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13522063@std.stei.itb.ac.id

***Abstract*—Kesalahan dalam penulisan bisa menyebabkan misinterpretasi informasi dan mengurangi kredibilitas teks. Hal ini sangat krusial dalam konteks akademis, hukum, dan publikasi profesional, di mana keakuratan informasi adalah prioritas utama. Untuk mengatasi masalah ini, pengembangan *typo checker* yang efektif menjadi sangat penting. Pembuatan *typo checker* yang dijelaskan dalam makalah ini mengimplementasikan algoritma *string matching* seperti Boyer-Moore, Knuth-Morris-Pratt, Regular Expression, dan Levenshtein Distance. Tiap-tiap algoritma ini memiliki keunggulan dalam mendeteksi jenis kesalahan tertentu, sehingga kombinasi mereka diharapkan dapat menawarkan solusi komprehensif dan efisien dalam deteksi kesalahan penulisan. Diharapkan, metodologi yang diajukan dalam makalah ini tidak hanya membantu dalam deteksi kesalahan penulisan tetapi juga dalam memperbaiki kesalahan tersebut secara otomatis, sehingga dapat meningkatkan efisiensi proses editing dan review dokumen.**

Keywords—String Matching; Typographical Error; Algorithm

I. Pendahuluan

Akurasi dan efisiensi pengolahan teks merupakan komponen yang penting dalam penulisan suatu dokumen. *Typographical error* (*typo*) dan inkonsistensi dalam penulisan dapat menyebabkan misinterpretasi informasi dan mengurangi kredibilitas dari teks. Kesalahan dalam penulisan disebabkan oleh kesalahan dalam pengetikan sehingga menghasilkan ejaan kata yang tidak bermakna. Kesalahan ini bisa menyebabkan pembaca untuk salah mengartikan maksud dari suatu tulisan. Berdasarkan permasalahan tersebut, diperlukan solusi untuk bisa mendeteksi dan mengoreksi kesalahan dengan cepat dan tepat. Untuk bisa menyelesaikan permasalahan tersebut, akan dianalisis penggunaan algoritma *string matching* sebagai solusinya.

Makalah ini bertujuan untuk mengeksplorasi aplikasi algoritma *string matching* dalam mendeteksi kesalahan ketik serta melakukan identifikasi kata yang perlu dikoreksi. Melalui penulisan makalah ini, penulis akan menganalisis penggunaan algoritma *string matching* seperti Knuth-Morris-Pratt, Boyer-Moore, dan *Regular Expression* (*Regex*) untuk mengidentifikasi kesalahan penulisan serta penggunaan *Levenshtein Distance* untuk mencari kata koreksi yang paling dekat.

Diharapkan, melalui penerapan algoritma string matching, proses pengecekan typo dan perbaikan teks dapat menjadi lebih otomatis, akurat, dan efisien, sehingga dapat meminimalisir kesalahan dalam dokumen-dokumen penting. Melalui makalah ini, pembaca diharapkan dapat memperoleh pemahaman yang lebih mendalam tentang potensi dan kegunaan algoritma *string matching* dalam konteks ini.

II. Dasar Teori

A. Algoritma

Algoritma adalah urutan langkah-langkah untuk menyelesaikan suatu persoalan, dengan memproses masukan menjadi luaran. Analisis algoritma adalah metode pengukuran kinerja (*performance*) algoritma dari segi efisiensinya. Maka, dalam algoritma, parameter utama dari penggunaan suatu algoritma adalah keberhasilan suatu algoritma dalam menyelesaikan masalah, seberapa efisien algoritma tersebut dalam memperoleh solusi, seberapa banyak *resource* yang diperlukan untuk menjalankan algoritma tersebut. Secara umum, parameter tersebut dibagi menjadi:

1. Kompleksitas waktu (T(n))
   Jumlah tahapan komputasi (operasi-operasi yang terdapat pada algoritma) yang dilakukan di dalam algoritma sebagai fungsi dari ukuran masukan n.
2. Kompleksitas ruang (S(n))
   Ruang memori yang dibutuhkan algoritma sebagai fungsi dari ukuran masukan n.

Strategi algoritma adalah pendekatan umum untuk memecahkan persoalan secara algoritmik sehingga dapat diterapkan pada bermacam-macam persoalan dari berbagai bidang komputasi. Beberapa contoh strategi algoritma adalah sebagai berikut:

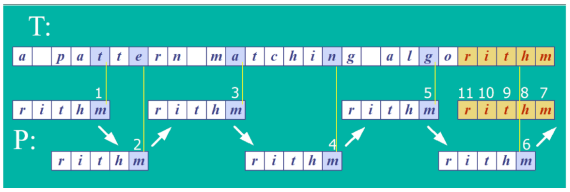
1. Algoritma *Brute-Force*
2. Algoritma *Greedy*
3. Algoritma *Divide and Conquer*
4. Algoritma *Decrease and Conquer*
5. Algoritma *Backtracking*
6. Algoritma *Branch and Bound*
7. Algoritma *String Matching*
8. *Dynamic Programming*

B. Algoritma String Matching

Algoritma string matching adalah algoritma yang bertujuan untuk mencari pattern tertentu pada suatu teks. Misalkan S adalah sebuah string dengan panjang m, maka substring S[i..j] dari S adalah fragmen string yang terletak antara indeks i dan j. Terdapat empat algoritma yang akan digunakan untuk mencocokkan string pada permasalahan ini:

1. Algoritma Boyer-Moore

   Algoritma Boyer-Moore (BM) merupakan algoritma pencocokan string yang memanfaatkan dua teknik. Algoritma ini memproses pola yang dicari, bukan teks yang dicari polanya.



Gambar 1. Contoh Ilustrasi Kasus Boyer-Moore
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

   Pada algoritma ini terdapat dua aturan utama dalam pergeseran pencarian pola.

   - *Bad Character Heuristic*

     Aturan ini mempertimbangkan karakter dalam teks (T) yang proses perbandingannya gagal (dengan asumsi terjadi ketidakcocokan). Jika kemunculan karakter tersebut terdapat di sebelah kiri pola (P), dilakukan pergeseran agar kemunculan karakter tersebut sejajar dengan karakter yang proses perbandingannya gagal. Jika karakter tidak muncul di sebelah kiri P, maka pergeseran terjadi sepanjang P, melewatkan lokasi dimana perbandingan gagal.

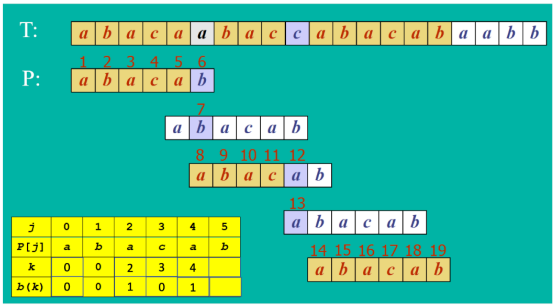
   - *Good Suffix Heuristic*

     Aturan ini menggunakan fitur membandingkan algoritma mulai dari akhir pola dan bergerak menuju awal. Misalkan t adalah substring dari teks T yang cocok dengan substring dari pola P. Pergeseran pola dilakukan dalam kasus berikut:

     - Kemunculan t lainnya di dalam P cocok dengan t di dalam T
     - Awalan P cocok dengan akhiran t
     - P bergerak melewati t karena tidak ada kejadian lebih lanjut pola yang ditemukan pada pencocokan sebelumnya

2. Algoritma Knuth-Morris-Pratt

   Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma pencocokan string yang mencari kemunculan suatu pola atau *substring* dalam teks dari kiri ke kanan. Algoritma ini berperilaku seperti algoritma *brute force* tetapi dengan pendekatan pemindaian pola yang lebih efisien dibandingkan dengan brute force.



Gambar 2. Contoh Ilustrasi Kasus Knuth-Morris-Pratt
Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

3. Regular Expression

   Regular expression (regex) adalah notasi standar yang mendeskripsikan suatu pola (*pattern*) berupa urutan karakter atau string. Regex digunakan untuk pencocokan string (*string matching*) dengan efisien.
   Regex untuk *character class*:

Tabel 1. Character Class

Construct	**Deskripsi**
`[abc]`	a, b, atau c (single class)
`[^abc]`	semua karakter selain a, b, dan c (negasi)
`[a-zA-Z]`	a sampai z atau A sampai Z, inclusive (range)
`[a-d[m-p]]`	a sampai d atau m sampai p (gabungan)
`[a-z&&[def]]`	d, e atau f (irisan)
`[a-z&&[^bc]]`	a sampai z, kecuali b dan c (subtraksi)
`[a-z&&[^m-p]]`	a sampai z, dan bukan m sampai p (subtraksi)

Regex untuk *character class* yang *predefined*:

Tabel 2. Predefined Character Class

Construct	**Deskripsi**
`.`	semua karakter
`\d`	Digit [0-9]
`\D`	Non Digit [^0-9]
`\s`	karakter whitespace

\S	karakter non whitespace
\w	word character
\W	non word character

Boundary matchers:

Tabel 3. Boundary Matchers

Construct	Deskripsi
^	awal baris
$	akhir baris
\b	batas kata
\B	batas bukan kata
\G	akhir match sebelumnya
\Z	akhir dari input tapi untuk final terminator jika ada
\z	akhir dari input

4. Levenshtein Distance

Levenshtein distance adalah metode pengukuran perbedaan antara dua string. Semakin besar jaraknya maka semakin besar ketidakmiripan kedua string tersebut. Jarak Levenshtein dihitung menggunakan persamaan berikut:

$$\text{lev}(a,b) = \begin{cases} |a| & \text{jika } |b| = 0, \\ |b| & \text{jika } |a| = 0, \\ \text{lev}(\text{tail}(a), \text{tail}(b)) & \text{jika } a[0] = b[0] \\ 1 + \min \begin{cases} \text{lev}(\text{tail}(a), b) \\ \text{lev}(a, \text{tail}(b)) \\ \text{lev}(\text{tail}(a), \text{tail}(b)) \end{cases} & \text{lainnya.} \end{cases}$$

C. Kesalahan Penulisan (Typo)

Typographical Error atau *typo* adalah kesalahan yang dilakukan pada proses pengetikan sehingga terjadi kesalahan ejaan pada suatu kata tertentu. Umumnya, *typo* dapat terjadi karena tangan melakukan *slip* ketika mengetik. Beberapa jenis *typo* yang umum terjadi adalah sebagai berikut:

1. *Deletion*
   Kekurangan huruf pada suatu kata. Contoh, kata ‘tidak’ diketik sebagai ‘tdk’.
2. *Substitution*
   Huruf yang salah pada suatu kata. Contoh, kata ‘tidak’ diketik sebagai ‘tidsk’.
3. *Transposition*
   Seluruh huruf pada suatu kata lengkap, namun terdapat beberapa posisi huruf yang tertukar atau salah. Contoh, kata ‘tidak’ diketik sebagai ‘tiadk’.

III. Penerapan Algoritma String Matching dalam Deteksi Kesalahan Penulisan dan Identifikasi Kata Koreksi

A. Preprocessing

Analisis kesalahan ketikan (typo) dilakukan per kata, sehingga teks masukan perlu dipecah menjadi kata-kata. Hal ini dilakukan dengan menggunakan *regular expression* sebagai berikut:

```
\b\w+\b
```

Contoh penggunaannya:

Tabel 4. Contoh Pemisahan Kata

Input	i notced an unusal shop that had just opend. Its windows were filled with al sorts of brighly colord objects that caught my attetion imediately.
Output	['i', 'notced', 'an', 'unusal', 'shop', 'that', 'had', 'just', 'opend', 'Its', 'windows', 'were', 'filled', 'with', 'al', 'sorts', 'of', 'brighly', 'colord', 'objects', 'that', 'caught', 'my', 'attetion', 'imediately']

Analisis kesalahan pada kapitalisasi di depan kalimat dilakukan per kalimat, sehingga teks masukan perlu dipecah menjadi kalimat-kalimat. Hal ini dilakukan dengan menggunakan regular expression sebagai berikut:

```
(?<!\w\.\w.)(?<![A-Z][a-z]\.)(?<=\.|\?)\s
```

Contoh penggunaan:

Tabel 5. Contoh Pemisahan Kalimat

Input	i notced an unusal shop that had just opend. Its windows were filled with al sorts of brighly colord objects that caught my attetion imediately.
Output	['i notced an unusal shop that had just opend.', 'Its windows were filled with al sorts of brighly colord objects that caught my attetion imediately.']

Perlu juga dilakukan pemuatan kamus yang berisi kata-kata yang benar. Kamus disimpan dalam suatu file text dengan format .txt. Hal ini dilakukan dengan melakukan hal berikut:

```
Function LoadDictionary(filepath)
    Initialize dictionary as an empty list

    Open the file at file path for reading as
file
        For each line in file
            Strip whitespace from the ends of
line
            Convert line to uppercase
            Add the line to dictionary

    Close the file

    Return dictionary
EndFunction
```

B. Identifikasi Typo

Untuk mengidentifikasi kesalahan penulisan (typo), digunakan algoritma KMP dan juga BM. Untuk semua kata pada kamus, kata tersebut dijadikan pola dan kata pada teks dijadikan teks penuhnya. Jika kata ditemukan menggunakan KM ataupun BM, maka kata tersebut diidentifikasi sebagai kata baku yang terdapat pada kamus.

```
Function IsValidWord(word, dictionary, stats)
    For each pattern in dictionary
        If KmpSearch(word, pattern, stats) OR
BmSearch(word, pattern, stats)
            If length of pattern equals length
of word
                Return True
            Else
                Continue to the next iteration
    Return False
EndFunction
```

Pencarian menggunakan KMP dimulai dengan mencari *longest prefix* yang juga merupakan *suffix*, kemudian membandingkan karakter per karakter antara kata dan pola, dan menggunakan untuk menghindari perbandingan ulang yang tidak perlu.

```
Function ComputeLps(pattern)
    Initialize lps array of size equal to length
of pattern with all zeroes
    Initialize length to 0
    Initialize i to 1
    While i is less than the length of the
pattern
        If pattern[i] equals pattern[length]
            Increment length
            lps[i] is set to length
            Increment i
        Else
            If length is not zero
                  Set length to lps[length - 1]
            Else
                  Set lps[i] to 0
                Increment i
     Return lps
EndFunction

Function KmpSearch(word, pattern, stats)
    Start timing
    Call ComputeLps with pattern to get lps
array
    Initialize i and j to 0
    While i is less than the length of the word
        Increment kmp_checks in stats
        If pattern[j] equals word[i]
            Increment i and j
        If j equals the length of the pattern
            Update kmp_time in stats with
elapsed time
            Return True
        Else If i is less than length of the
word and pattern[j] not equals word[i]
            If j is not zero
                Set j to lps[j - 1]
            Else
                Increment i
    Update kmp_time in stats with elapsed time
    Return False
EndFunction
```

Pencarian menggunakan BM dimulai dengan mencari heuristik "bad character" untuk melompati karakter yang tidak cocok tanpa perlu memeriksa setiap karakter pada setiap langkah, mengiterasi karakter-karakter dalam pola dari kanan ke kiri, jika karakter tidak cocok, fungsi melompat ke depan berdasarkan heuristik.

```
Function BmSearch(word, pattern, stats)
    Start timing
    If length of pattern is greater than length
of word
        Return False
    Call BadCharHeuristic with pattern to get
bad_char array
    Initialize m to length of pattern and n to
length of word
    Initialize s to 0
    While s is less than or equal to n - m
        Initialize j to m - 1
        While j is greater than or equal to 0
and pattern[j] equals word[s + j]
            Decrement j
            Increment bm_checks in stats
        If j is less than 0
            Update bm_time in stats with elapsed
time
            Return True
        Else
            Set s to maximum of 1 or j -
bad_char[ord(word[s + j])]
    Update bm_time in stats with elapsed time
    Return False
EndFunction
Function BadCharHeuristic(pattern)
    Initialize bad_char array of size 256 with
all values set to -1
    For each index i from 0 to the length of
pattern - 1
        Set bad_char[ord(pattern[i])] to i
    Return bad_char
EndFunction
```

C. Perbaikan Typo

Perbaikan kesalahan penulisan dilakukan dengan mencari kata terdekat menggunakan algoritma *Levenshtein Distance*. Untuk setiap kata dalam kamus, fungsi menghitung jarak Levenshtein antara kata yang diberikan dan kata dalam kamus menggunakan fungsi levenshtein. Jika jarak yang dihitung lebih kecil dari jarak terdekat yang telah tersimpan, maka jarak terdekat akan diperbarui dengan jarak baru dan kata terdekat akan diperbarui dengan kata tersebut.

```
Function LevenshteinDistance(s1, s2)
    If length of s1 is less than length of s2
        Return LevenshteinDistance(s2, s1)

    If length of s2 is zero
        Return length of s1

    Initialize previous_row as a range from 0 to
length of s2 + 1

    For each character index i and character c1
in s1
        Initialize current_row with first
element as i + 1

        For each character index j and character
c2 in s2
            Set insertions to previous_row[j +
1] + 1
            Set deletions to current_row[j] + 1
            If c1 equals c2
                Set substitutions to
previous_row[j]
            Else
                Set substitutions to
previous_row[j] + 1

            Append minimum of insertions,
deletions, and substitutions to current_row

        Set previous_row to current_row

    Return last element of previous_row
EndFunction

Function FindClosestWord(word, dictionary)
    Initialize min_distance to a very large
value (maxsize)
    Initialize closest_word to an empty string

    For each dict_word in dictionary
        Calculate distance using
LevenshteinDistance between word and dict_word
        If distance is less than min_distance
            Set min_distance to distance
            Set closest_word to dict_word

    Return closest_word and min_distance
EndFunction
```

D. Identifikasi Kesalahan Capitalization

Setelah kalimat dipisah-pisah, setiap kalimat akan dicek apakah memenuhi aturan (kalimat dimulai dengan kata dengan awalan huruf kapital).

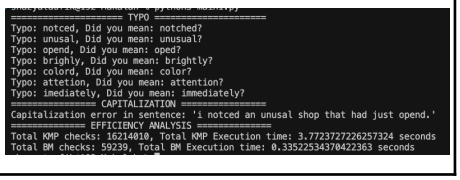
```
For each sentence in sentences
    If the first character of sentence is not
uppercase
        Print "Capitalization error in sentence:
'trimmed sentence'"
EndFor
```

IV. Analisis Hasil

Untuk menentukan efisiensi dari algoritma, maka perlu dilakukan analisis hasil dengan menjalankan program pada situasi-situasi tertentu.

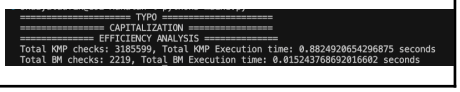
1. Test Case 1

Tabel 6. Test Case 1

Input teks	i notced an unusal shop that had just opend. Its windows were filled with al sorts of brighly colord objects that caught my attetion imediately.
Output	

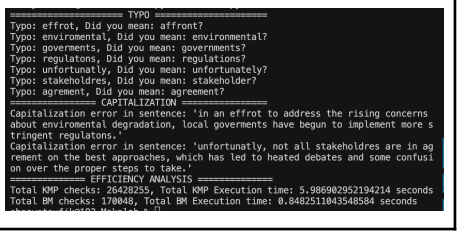
2. Test Case 2

Tabel 7. Test Case 2

Input teks	The quick brown fox jumps over the lazy dog.
Output	

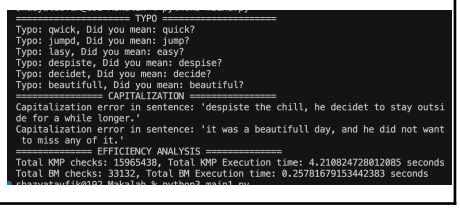
3. Test Case 3

Tabel 8. Test Case 3

Input teks	in an effrot to address the rising concerns about enviromental degradation, local goverments have begun to implement more stringent regulatons. unfortunatly, not all stakeholdres are in agrement on the best approaches, which has led to heated debates and some confusion over the proper steps to take.
Output	

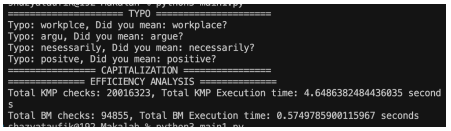
4. Test Case 4

Tabel 9. Test Case 4

Input teks	The qwick brown fox jumpd over the lasy dog. despiste the chill, he decidet to stay outside for a while longer. it was a beautifull day, and he did not want to miss any of it.
Output	

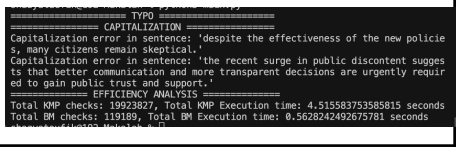
5. Test Case 5

Tabel 10. Test Case 5

Input teks	Many believe that the advance in technology, especially in robotics, is leading to significant changes in the workplce. However, some argu that these developments may not nesessarily bring about the positve outcomes that many anticipate.
Output	

6. Test Case 6

Tabel 11. Test Case 6

Input teks	despite the effectiveness of the new policies, many citizens remain skeptical. the recent surge in public discontent suggests that better communication and more transparent decisions are urgently required to gain public trust and support.
Output	

Berdasarkan hasil pengujian yang telah dilakukan oleh penulis, ada beberapa hal yang bisa diamati. Hal-hal tersebut adalah sebagai berikut:

1. Identifikasi typo dapat dilakukan dengan memanfaatkan algoritma string matching, yaitu Boyer-Moore dan juga Knuth-Morris-Pratt.
2. Levenshtein distance dapat menemukan kata terdekat dari typo. Namun, kata terdekat tersebut memiliki kemungkinan bahwa kata tersebut bukanlah kata yang dimaksud. Jadi, pencarian kata terdekat ini hanya mencari berdasarkan jarak tanpa mempertimbangkan maknanya.
3. Regular expression dapat secara efektif digunakan untuk memisahkan kata dan juga kalimat.
4. Jumlah karakter yang dibandingkan pada algoritma KMP selalu lebih banyak daripada jumlah karakter yang dibandingkan pada algoritma BM.
5. Waktu eksekusi algoritma BM selalu lebih cepat daripada KMP.

Beberapa hal yang bisa disimpulkan berdasarkan hal tersebut adalah bahwa algoritma string matching, seperti BM, KMP, dan regex terbukti dapat menyelesaikan permasalahan kesalahan penulisan. BM lebih efisien dibandingkan KMP, hal ini disimpulkan dari banyaknya karakter yang masing-masing algoritma bandingkan dan juga waktu eksekusinya.

V. Kesimpulan

Berdasarkan analisis yang telah dilakukan, dapat disimpulkan bahwa, algoritma Boyer-Moore (BM) dan Knuth-Morris-Pratt (KMP) efektif dalam mengidentifikasi typo dalam teks. Meskipun kedua algoritma tersebut efektif, BM menunjukkan efisiensi yang lebih tinggi dibandingkan dengan KMP dari segi jumlah karakter yang dibandingkan dan waktu eksekusi. Hal ini menunjukkan bahwa BM dapat menjadi pilihan yang lebih baik dalam situasi yang memerlukan pemrosesan cepat dan efisien atas volume data yang besar.

Fungsi jarak Levenshtein terbukti berguna untuk menemukan kata terdekat dari sebuah typo, meskipun kata yang ditemukan belum tentu merupakan kata yang dimaksud penulis. Regular expression (regex) terbukti sangat efektif dalam memisahkan kata dan kalimat, membantu dalam proses analisis teks lanjutan dan memungkinkan algoritma lain untuk bekerja pada unit teks yang lebih spesifik.

Penggabungan BM, KMP, dan regex dalam pembangunan sistem typo checker menyediakan solusi yang robust untuk mendeteksi kesalahan penulisan. Namun, penambahan analisis semantik mungkin diperlukan untuk meningkatkan relevansi saran kata yang diberikan oleh sistem berbasis jarak seperti Levenshtein.

Meskipun algoritma-algoritma ini sangat berguna dalam pengidentifikasian dan koreksi kesalahan penulisan, masih ada ruang untuk penyempurnaan, khususnya dalam aspek pemahaman konteks dan makna kata untuk memastikan bahwa saran yang diberikan tidak hanya berdekatan secara leksikal tetapi juga relevan secara semantik.

Link Video Youtube

https://youtu.be/dsn7vRSyDAE

Link Kode Program

https://drive.google.com/drive/folders/1RMO8r1_sYFW-1piBIHVWsL3cIjCMdXE4?usp=sharing

Ucapan Terima Kasih

Puji syukur kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya, penulis dapat menyelesaikan makalah yang berjudul ‘Penerapan Algoritma String Matching dalam Deteksi Kesalahan Penulisan dan Identifikasi Kata Koreksi’ ini dengan baik dan lancar. Penulis juga ingin berterima kasih kepada tim pengajar IF2211 - Strategi Algoritma, terutama dosen K01, Dr. Ir. Rinaldi Munir, M.T., karena atas ilmu serta sumber materi yang diajarkannya di kelas selama perkuliahan dapat membantu penulis untuk menyelesaikan makalah ini. Penulis juga ingin berterima kasih kepada teman-teman penulis serta keluarga penulis yang telah membantu penulis dalam memilih topik penelitian makalah ini dan juga memberikan dukungan dalam proses penyelesaian makalah ini.

References

[1] Munir, Rinaldi. 2024. “Pengantar Strategi Algoritma” https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/Pengantar-Strategi-Algoritma-(2024).pdf (Diakses 10 Juni 2024)
[2] Munir, Rinaldi. 2021. “Pencocokan String (String/Pattern Matching)” https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf (Diakses 10 Juni 2024)
[3] Munir, Rinaldi. 2023. “String Matching dengan Regular Expression” https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/String-Matching-dengan-Regex-2019.pdf (Diakses 10 Juni 2024)
[4] Khodra, Masayu Leylia; Wibisono, Yudi. 2020. “Modul Praktikum Kuliah Pengantar Regular Expression” https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Modul-Praktikum-NLP-Regex.pdf (Diakses 10 Juni 2024)
[5] Fahma, Arina Indana. 2018. “Identifikasi Kesalahan Penulisan Kata (*Typographical Error*) pada Dokumen Berbahasa Indonesia Menggunakan Metode N-gram dan *Levenshtein Distance*” https://www.researchgate.net/publication/323365722_Identifikasi_Kesalahan_Penulisan_Kata_Typographical_Error_pada_Dokumen_Berbahasa_Indonesia_Menggunakan_Metode_N-gram_dan_Levenshtein_Distance (Diakses 10 Juni 2024)

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, 12 Juni 2024

Shazya Audrea Taufik
13522063