# Aplikasi Pencocokan *String* dalam Otomatisasi Penulisan Miring dalam Dokumen Berbahasa Indonesia

Jeremia Axel 13519188
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13519188@std.stei.itb.ac.id

***Abstract*—Dalam pedoman umum ejaan bahasa Indonesia (PUEBI), kalimat atau kata bahasa lain harus ditik miring (*italic*) atau diberi garis bawah (*underline*) jika ditulis tangan. Sering dalam membuat karya ilmiah atau dokumen lainnya, penulis memasukan kata atau frasa bahasa lain seperti bahasa Inggris karena tidak terdapat padanan kata dalam bahasa Indonesia atau jika diterjemahkan akan kehilangan maknanya. Memiringkan penulisan adalah perkara sederhana namun sering merepotkan. Dengan menggunakan pencocokan *string* atau *string matching*, program dapat mengotomatisasi penulisan miring sehingga penulis dapat lebih fokus pada konten dibanding disibukan oleh hal-hal teknis.**

***Keywords—Pencocokan string; penulisan miring; dokumen;***

## I. PENDAHULUAN

Dalam menulis dokumen atau karya ilmiah yang baik dan benar sesuai dengan pedoman umum ejaan bahasa Indonesia (PUEBI), penulis diharuskan memiringkan kata atau frasa bahasa lain. Penulisan miring (*italic*) berguna memberitahu pembaca bahwa kata tersebut berasal dari bahasa lain yang memiliki makna khusus.

Penulisan miring merupakan hal yang sederhana namun cukup merepotkan karena penulis harus memindai dokumen dari awal sampai akhir (atau awal sampai akhir suatu partisi) untuk mengubah kata atau frasa bahasa lain. Setelah itu, penulis harus sekali lagi memindai atau membaca ulang untuk memastikan tidak terdapat kata atau frasa yang terlewat.

Cara lain adalah dengan sembari menulis atau mengetik, memastikan tiap kata atau frasanya sudah tepat. Namun hal ini juga cukup merepotkan karena harus mengubah-ubah jenis tulisan berulang kali. Setelah dokumen atau bagian tersebut ditulis, penulis tetap harus membaca ulang untuk memastikan tidak ada kata atau frasa yang terlewat.

Karena itu, untuk memudahkan penulisan, dapat digunakan program yang dapat memindai dan mengubah kata atau frasa dalam bahasa lain menjadi miring (*italic*). Untuk itu, dapat digunakan algoritma pencocokan *string* untuk memindai setiap kata atau frasa dan memeriksa apakah terdapat kata atau frasa dari bahasa lain dan kemudian memiringkannya.

## II. DASAR TEORI

### A. *String*

Dalam ilmu komputer, *string* adalah sekumpulan karakter terurut secara linear. Umumnya, *string* diapit oleh dua tanda kutip [3]. Contoh *string* sederhana adalah “Halo, Dunia!”

*String* memiliki banyak kegunaan, misalnya setiap masukan dari papan tik merupakan *string*. *String* tersebut kemudian dikonversi menjadi tipe data yang diperlukan, jika angka maka akan dikonvesi menjadi angka yang berkorespondensi. Maka “120” dan 120 merupakan dua hal yang berbeda. Sebuah dokumen juga umumnya disimpan menjadi sebuah *string* panjang yang tiap barisnya diakhiri dengan karakter ‘\n’ yang berarti baris baru atau *new line*.

TABLE I. CONTOH *STRING*

| **Dalam teks biasa :** |
|---|
| Dalam ilmu komputer, *string* adalah sekumpulan karakter terurut secara linear. Umumnya, *string* diapit oleh dua tanda kutip. Contoh *string* sederhana adalah “Halo, Dunia!” |
| **Dalam representasi program :** |
| “Dalam ilmu komputer, *string* adalah sekumpulan karakter\n terurut secara linear. Umumnya, *string* diapit oleh dua\n tanda kutip. Contoh *string* sederhana adalah \“Halo, Dunia!\”“ |

Figur 1. Teks umumnya terlihat dengan representasinya dalam pemrograman.

Sebuah *string* memiliki beberapa properti seperti panjang *string*, prefiks, dan sufiks. Misalnya sebuah *string* dengan panjang *m*, $S = x_0x_1\ldots x_{m-1}$, memiliki prefiks (*prefix*) *S[0..k]* dan sufiks (*suffix*) *S[k..m-1]* dengan k merupakan indeks dari *string* antara 0 dan *m-1*. Misalnya:

- S : “HELLO”
- Prefiks : “H”, “HE”, “HEL”, “HELL”, “HELLO”
- Sufiks : “O”, “LO”, “LLO”, “ELLO”, “HELLO”

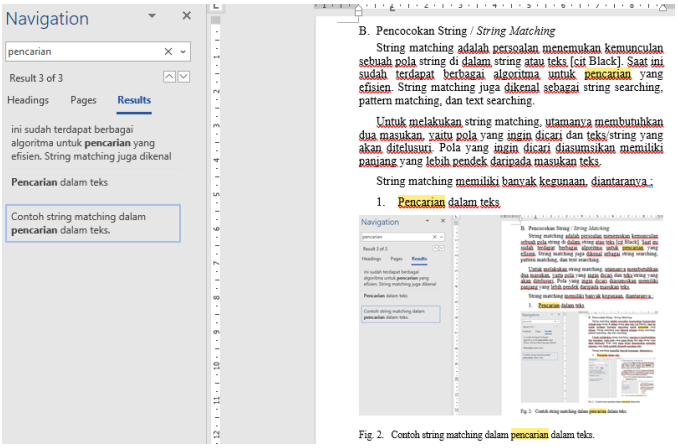
B. Pencocokan *String* / *String Matching*

*String matching* adalah persoalan menemukan kemunculan sebuah pola *string* di dalam *string* atau teks [6]. Saat ini sudah terdapat berbagai algoritma untuk pencarian yang efisien. *String* matching juga dikenal sebagai *string searching*, *pattern matching*, dan *text searching*.

Untuk melakukan *string matching*, utamanya membutuhkan dua masukan, yaitu pola yang ingin dicari dan teks/*string* yang akan ditelusuri. Pola yang ingin dicari diasumsikan memiliki panjang yang lebih pendek daripada masukan teks.

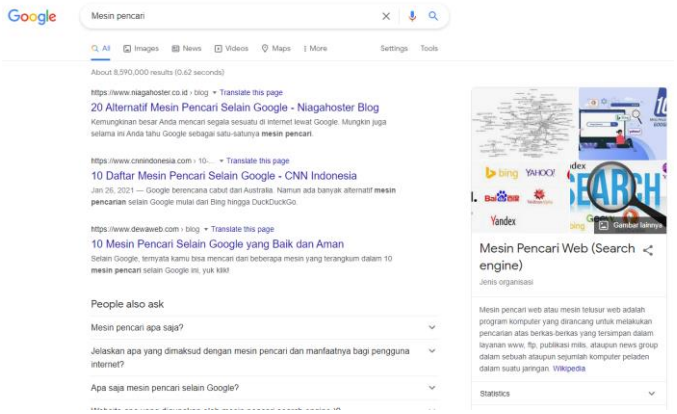
*String matching* memiliki banyak kegunaan dalam kehidupan sehari-hari, diantaranya :

1. Pencarian dalam teks




Figur 2. Contoh *string matching* dalam pencarian dalam teks.

2. Mesin pencari web




Figur 3. Contoh *string matching* dalam mesin pencari.

3. Analisis citra
4. Bioinformatika

C. Algoritma Pencocokan *String*

Dalam melakukan pencocokan *string*, terdapat beragam algoritma, yaitu Brute Force, Knuth-Morris-Pratt, Boyer-Moore, dan berbagai pengembangan lainnya.

1. *Brute Force*

*Brute Force* merupakan algoritma paling sederhana. Pola yang ingin dicari dicocokan satu persatu dari awal teks sampai akhir. Kelebihannya adalah kesederhanaannya sehingga mudah mengimplementasikan dan mudah memahaminya. Namun algoritma ini merupakan algoritma yang "mahal" karena terkadang memeriksa karakter yang tidak perlu.

Algoritma ini cocok untuk masukan teks dengan variasi karakter yang banyak (teks biasa). Namun jika masukan teks memiliki variasi yang sedikit (teks biner, gambar), akan membutuhkan waktu yang cukup lama. Kasus terburuknya *O(mn)*, sedangkan kasus terbaiknya *O(n)* dan kasus reratanya *O(m+n)* dengan *m* adalah panjang pola dan *n* adalah panjang teks.

2. Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt (KMP) merupakan algoritma pencocokan *string* seperti Brute Force namun memanfaatkan prefiks dan sufiks pada konsep *string*. Dengan memanfaatkan konsep tersebut, KMP bisa melakukan pergeseran pola dengan lebih efisien dibandingkan *Brute Force*.

Algoritma KMP, melakukan prapemrosesan (*preprocessing*) pada pola untuk mendapatkan indeks pencocokan pola yang baru jika pada suatu karakter pada pola terjadi ketidakcocokan dengan karakter di teks.

P (Pola) : "ABAACB"

| j | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| P[j] | A | B | A | A | C | B |
| b(j-1) | - | 0 | 0 | 1 | 1 | 0 |

T (Teks) : "ABABACABAACBA"

| A | B | A | B | A | C | A | B | A | A | C | B | A |
|---|---|---|---|---|---|---|---|---|---|---|---|---|

| A | B | A | A | C | B |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

| A | B | A | A | C | B |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

| A | B | A | A | C | B |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

| A | B | A | A | C | B |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

| A | B | A | A | C | B |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

Figur 4. Contoh pencocokan *string* dengan algoritma Knuth-Morris-Pratt. Warna hijau menunjukan teks masukan. Warna merah menunjukan karakter ketika tidak cocok dengan karakter di teks. Warna biru menunjukan karakter cocok.

Prapemrosesan pola dilakukan dengan memberi nilai untuk setiap j (karakter ke j dari pola) yaitu panjang maksimal prefiks P[0..j-1] yang juga sufiks P[1..j-1] dengan j dari 0 sampai panjang pola - 1.

Pada contoh, jika ketidakcocokan ditemukan pada karakter di indeks 3 (pada pola), pencocokan yang berikutnya, pola dimulai dari indeks 1. Karena P[0..3] ("A".."ABAA") dan P[1..3] ("A".. "BAA") terpanjang adalah "A" (1).

Algoritma ini cocok untuk teks yang sangat besar karena pencocokan tidak pernah bergerak mundur [1]. Namun algoritma ini tidak cocok untuk teks yang memiliki variasi karakter yang besar. Semakin besar maka semakin besar kemungkinan ketidakcocokan yang pada akhirnya akan menjadi sama dengan algoritma *Brute Force*. Rata-rata kompleksitasnya *O(m)* untuk prapemrosesan dan *O(n)* untuk pencocokan *string* sehingga *O(m+n)*.

3. Boyer-Moore

Algoritma Boyer-Moore menggunakan dua teknik, yaitu:

a. *Looking-Glass*

Berbeda dari kedua algoritma sebelumnya, algoritma ini melakukan mencocokan karakter pada pola dengan karakter pada teks dimulai dari karakter paling akhir pada pola dan bergerak ke kiri. Meskipun demikian, pergeseran pola tetap dari kiri ke kanan.

b. *Character-Jump*

Pada algoritma Boyer-Moore, pelompatan karakter yang dilakukan bervariasi berdasarkan kemunculan terakhir karakter-karakter dari teks di pola. Pada dasarnya terdapat tiga kasus :

**Kasus I**

Jika kemunculan terakhir karakter T[i] (karakter ke i pada teks) terdapat di sebelah kiri karakter P[j] (karakter ke j pada pola), geser pola ke kanan sehingga kemunculan karakter terakhir tersebut sama dengan T[i].

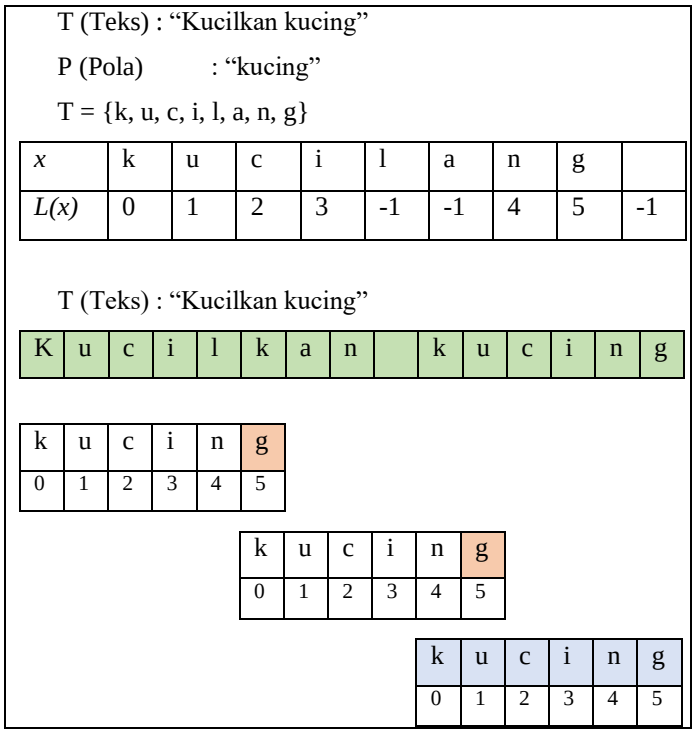
**Kasus II**

Jika kemunculan terakhir karakter T[i] terdapat di sebelah kanan karakter P[j] atau seperti kasus I namun tidak memungkinkan pergeseran pola sehingga karakter T[i] bersesuaian dengan karakter yang sama, geser pola ke kanan satu kali.

**Kasus III**

Jika tidak terdapat kemunculan terakhir karakter T[i] pada pola, maka geser pola sehingga awal pola (P[0]) dimulai dari karakter berikutnya pada teks (T[i+1]).

Algoritma Boyer-Moore, sama seperti algoritma Knuth-Morris-Pratt, melakukan prapemrosesan. Perbedaannya, pada algoritma Boyer-Moore, pemrosesan dilakukan dengan mengiterasi teks dan mencatat karakter apa saja yang terdapat pada teks. Kemudian mengiterasi pola dan mencatat indeks terakhir kemunculan karakter-karakter (dari teks) di pola. Umumnya kemunculan karakter-karakter ini dicatat pada sebuah *array*.




| *x* | k | u | c | i | l | a | n | g | |
|---|---|---|---|---|---|---|---|---|---|
| *L(x)* | 0 | 1 | 2 | 3 | -1 | -1 | 4 | 5 | -1 |

| K | u | c | i | l | k | a | n | | k | u | c | i | n | g |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

| k | u | c | i | n | g |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

| k | u | c | i | n | g |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

| k | u | c | i | n | g |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

Figur 5. Contoh pencocokan *string* dengan algoritma Boyer-Moore. Warna hijau menunjukan teks masukan. Warna merah menunjukan karakter ketika tidak cocok dengan karakter di teks. Warna biru menunjukan karakter cocok.

Algoritma Boyer-Moore, berkebalikan dengan algoritma Knuth-Morris-Pratt, cocok untuk masukan teks dengan variasi karakter yang banyak namun kurang cocok dengan teks dengan variasi karakter yang sedikit. Waktu yang diperlukan untuk prapemrosesan adalah *O(m+k)* untuk mengiterasi teks dan pola. Kompleksitas algoritma untuk kasus terburuknya adalah *O(mn)* (pola terdapat di teks) dan kasus terbaiknya adalah *O(n/m)* (panjang teks/panjang pola).

Algoritma Boyer-Moore merupakan algoritma yang banyak digunakan sebagai algoritma standar pencarian *string*. Hal ini karena kecocokannya dengan bahasa manusia khususnya bahasa Inggris.

4. *Regular Expression*

*Regular Expression*, yang disingkat Regex, adalah *string* yang mengambarkan *string* lain atau sekumpulan *string* [2]. *Regular Expression* memiliki sintaks khusus untuk dapat menggambarkan kombinasi dari susunan *string* lain.

Umumnya dalam bahasa pemrograman, diperlukan *library* atau modul atau *API* khusus yang menangani *regular expression*. Misalnya dalam Python3 perlu diimpor modul re untuk *regular expression*, dalam Java perlu diimpor modul java.util.regex, sedangkan dalam JavaScript tidak perlu diimpor tetapi menggunakan sintaks khusus (/../).

Berikut beberapa contoh sintaks dalam *regular expression* yang cukup sering digunakan dari referensi [5] :

TABLE II. SINTAKS REGULAR EXPRESSION

| Sintaks | Contoh | Penjelasan |
|---|---|---|
| . | H.i | . menyatakan karakter apapun selain baris baru, jadi contoh cocok dengan "Hai", "Hoi", "H1i", dan semacamnya. |
| \| | Kucing\|Anjing | \| menyatakan atau, jadi contoh akan cocok dengan *string* "Kucing" dan "Anjing" . |
| [ab] | [Kk]ucing | Karakter dalam kurung siku ([ ]) menyatakan semua karakter yang mungkin, jadi dalam contoh akan cocok dengan "Kucing" maupun "kucing". |
| [a-b] | [A-Z][a-z]* | Karakter dengan tanda minus dalam kurung siku menyatakan kemungkinan karakter dari karakter A sampai Z, jadi dalam contoh, *string* dimulai dengan satu huruf kapital dengan rentang A sampai Z dan sisanya diikuti huruf kecil (juga bisa tidak ada). |
| * | [A-Z][a-z]* | Menyatakan bahwa *string* dimulai dengan satu huruf kapital dan sisanya diikuti huruf kecil (juga bisa tidak ada). |
| + | [A-Z][a-z]+ | Menyatakan bahwa *string* dimulai dengan huruf kapital dan sisanya diikuti minimal satu huruf kecil. |

Figur 6. Beberapa contoh sintaks dalam *regular expression* yang cukup sering digunakan.

Kelebihan *regular expression* daripada *string matching* lain (*exact string matching*) adalah kemampuannya untuk mendeskripsikan kombinasi karakter lain hanya dengan sintaks yang relatif pendek. Jika menggunakan *exact string matching*, perlu mendefinisikan semua kemungkinan pola masing-masing satu pola lengkap. Hal ini merupakan hal yang sangat tidak efisien. Dengan adanya *regular expression*, pendeskripsian pola dapat lebih mudah dilakukan.

*Regular expression* umumnya dipakai untuk menjadi *filter* (penyaring) sebuah masukan *string*. Misalnya untuk memeriksa dan mengambil sebuah tanggal. Masukan tanggal dapat berupa format *dd-mm-yyyy* atau *dd/MM/YYYY* atau *day-month-yyyy*.

```
"((\d{1,2}(/|\s|-)\d{1,2}(/|\s|-)[\d]+)|(\d{1,2}(/|\s|-)((januari)|(februari)|(maret)|(april)|(mei)|(juni)|(juli)|(agustus)|(september)|(oktober)|(november)|(desember))(/|\s|-)\d{1,4}))"
```

D. Penulisan Dokumen

Dalam penulisan sebuah dokumen dalam bahasa Indonesia, ada beberapa aturan yang harus dipatuhi untuk menghasilkan dokumen yang baik dan benar. Aturan-aturan ini disebut Pedoman Umum Ejaan Bahasa Indonesia atau disingkat PUEBI. Aturan ini berguna untuk menghasilkan dokumen yang terstandar sehingga lebih mudah dan nyaman untuk dibaca serta meminimalkan kemungkinan misinterpretasi.

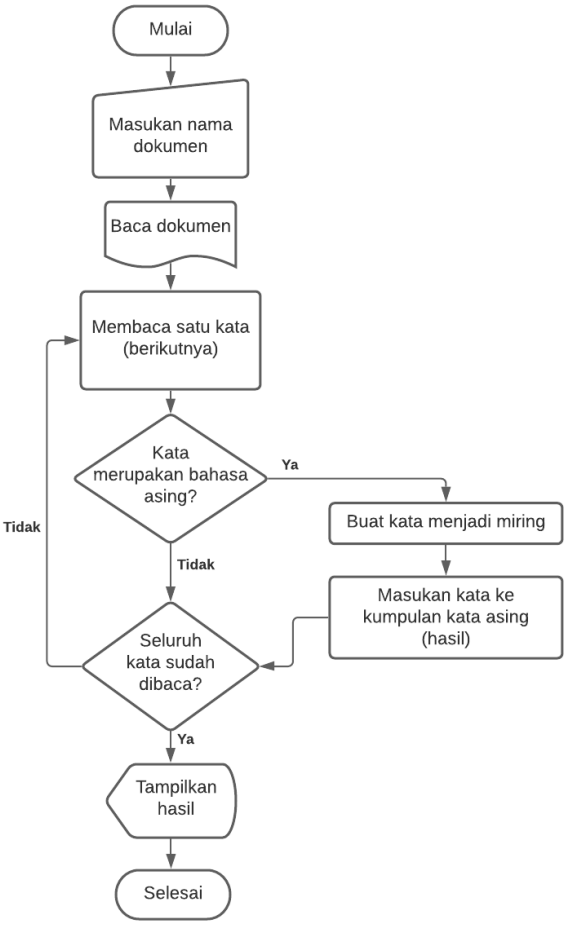
Salah satu aturan yang cukup sering dihadapi dalam penulisan sebuah dokumen adalah aturan penulisan miring. Menurut dokumentasi PUEBI [4], huruf miring digunakan untuk beberapa hal yaitu :

1. Menuliskan judul buku, nama majalah, atau nama surat kabar yang dikutip dalam tulisan, termasuk daftar pustaka.
2. Menegaskan atau menkhususkan huruf, bagian kata, kata, atau kelompok kata dalam kalimat.
3. Menuliskan kata atau ungkapan dalam bahasa daerah atau bahasa asing.

Dalam makalah ini akan lebih terfokus pada aturan yang ketiga, yaitu, menuliskan kata atau ungkapan bahasa asing. Menulis karya ilmiah akan sering membutuhkan referensi yang menggunakan bahasa asing, umumnya bahasa Inggris. Hal ini salah satunya disebabkan karena bahasa Inggris dianggap sebagai bahasa internasional.

## III. IMPLEMENTASI DAN PENGUJIAN

Rancangan algoritma program




Figur 7. Diagram alir rancangan algoritma program

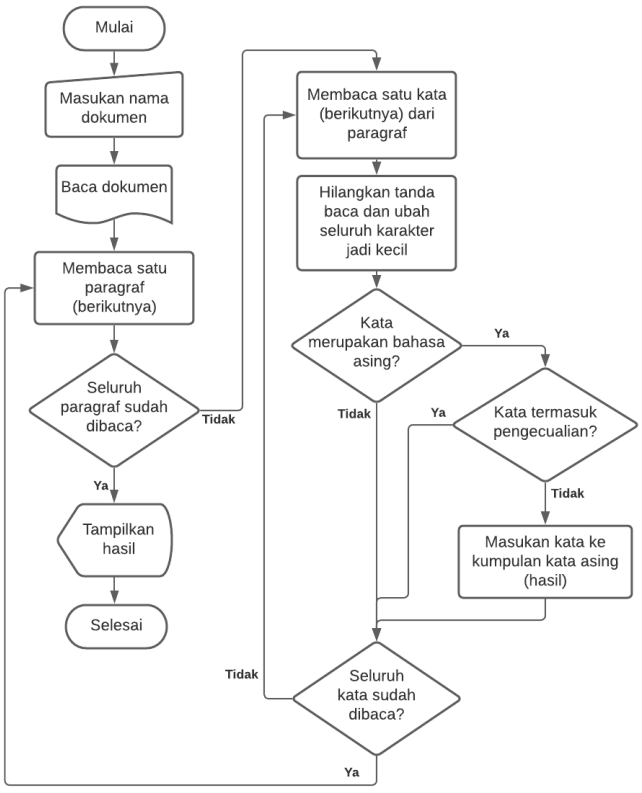
Implementasi dari rancangan ini menggunakan bahasa pemrograman Python3. Python3 dipilih karena kemudahan dan fleksibilitasnya. Dalam implementasi kali ini juga menggunakan bahasa Inggris sebagai bahasa asing yang digunakan.

Untuk dapat mendeteksi kata-kata yang berasal dari bahasa Inggris, perlu dibuat suatu kumpulan yang berisi kata-kata bahasa Inggris tersebut. Di Python dapat menggunakan *library NLTK Corpus* yang berisi kosakata-kosakata dari bahasa Inggris. Setelah dilakukan percobaan, ternyata *NLTK* memuat beberapa kata yang seharusnya tidak termasuk kata bahasa Inggris, karena itu dibuat pengecualian.

Selain itu, karena keterbatasan *library* yang ada dan juga keterbatasan penulis, program tidak dapat secara langsung mengubah sebuah kata menjadi miring. Karena itu, program dibuat menjadi menampilkan seluruh kata-kata berbahasa Inggris yang dapat ditemukan di teks.

Meskipun rancangan awal tidak tercapai, dengan bentuk program seperti ini, setidaknya penulis dokumen dapat lebih mudah karena dapat langsung mengetahui kosakata bahasa Inggris apa saja yang terdapat dalam teks. Kemudian, penulis dapat melakukan format ulang terhadap kata-kata tersebut menggunakan fitur bawaan dari pengedit teks.

Berikut algoritma keseluruhan program yang dapat terimplementasi secara lebih detail.




Figur 8. Diagram alir implementasi algoritma program secara detail

Misalkan sebuah dokumen terdiri dari *p* buah paragraf, dan setiap paragraf terdiri dari *h* huruf, maka kompleksitas algoritma untuk program ini adalah *O(hp)*.

```
import string
import re
import time

from docx import Document
from nltk.corpus import words, stopwords
foreign_dict = set(words.words())
stopword_list =
set(stopwords.words('indonesian'))
punctuations = string.punctuation
regex = "(O\(.\))|(T\[.\])"
exceptions = ["kiri", "kali", "tipe", "orang",
"tulisan", "surat", "baris", "strategi", "diagr
am", "coe", "wig", "daring", "data"]
for ex in exceptions:
    regex += "|(" + ex + ")"

def read_docx(filename: str):
    document = Document(filename)
    return document

def is_foreign_word(word):
    return word in foreign_dict

def is_stopword(word):
    return word in stopword_list
def is_exception(word):
    return re.search(regex, word) or
len(word.translate(str.maketrans('','',
string.punctuation))) == 1


if __name__ == "__main__":
    filename = input()
    document = read_docx(filename)

    time_start = time.time()
    foreign_words = set()
    for paragraph in document.paragraphs:
        for word in paragraph.text.split(" "):
            word_check =
word.translate(str.maketrans('','',
string.punctuation)).lower()
            if not is_exception(word.lower()):
                if is_foreign_word(word_check)
and not is_stopword(word_check):

foreign_words.add(word_check)

    time_end = time.time()

    for word in foreign_words:
        print(word)
    print("Elapsed time :
{time}".format(time=time_end-time_start))
```

Figur 9. Kode sumber yang diimplementasikan menggunakan bahasa pemrograman Python3.

Program menggunakan algoritma pencocokan *string regular expression*. *Regular expression* dipilih karena fleksibilitasnya yang bisa mencocokan dengan pola yang beragam.

Sebagai contoh, digunakan dokumen ini untuk dideteksi kata-kata yang berbahasa asing. Pada saat dokumen ini digunakan, terdapat 2779 kata. Jika dilakuan percobaan, didapat


Figur 10. Potongan gambar hasil percobaan dengan banyak kata (*hp*) = 2779

Dapat dilihat bahwa masih terdapat beberapa kata yang masih dianggap sebagai bahasa asing. Namun hal itu karena kata tersebut memang sudah diserap menjadi bahasa Indonesia namun diserap secara utuh sehingga tidak terdapat perbedaan.

Waktu yang diperlukan oleh program adalah 0.04 detik. Hal ini sudah sangat cepat akibat adanya algoritma *hash* untuk mencari keberadaan kata di sebuah *list*. Algoritma ini sudah menjadi algoritma bawaan dari bahasa pemrograman Python3.

## IV. Simpulan

Rancangan ini akan sangat memudahkan penulis-penulis. Implementasi yang berhasil dibuat dari rancangan ini memang sangat jauh dari sempurna. Implementasi saat ini hanya sangat sedikit memudahkan penulis. Jika rancangan dapat terimplementasi dengan sempurna, akan banyak penulis yang dimudahkan sehingga proses memastikan ungkapan atau kata bahasa asing dapat diotomatisasi.

Implementasi yang telah dibuat saat ini masih dapat dikembangkan lebih lanjut sehingga bisa sesuai dengan rancangan yang dibuat. Rancangan yang dibuat juga bisa dikembangkan dengan pendekatan. Misalnya, dibanding memeriksa apakah suatu kata bahasa asing, bisa diperiksa dahulu apakah suatu kata merupakan bahasa Indonesia sehingga tidak ada kata serapan yang terdeteksi sebagai kata asing.

## Pranala Video di Youtube

Video singkat penggambaran mengenai makalah ini dapat dilihat di https://youtu.be/yk4PH5xoIJE.

## Ucapan Terima Kasih

Terima kasih kepada Dr. Ir. Rinaldi Munir, MT. dan tim dosen pengajar mata kuliah IF2211 Strategi Algoritma dalam program studi Teknik Informatika ITB yang sudah mengajarkan mengenai konsep-konsep strategi algoritma. Dengan demikian penulis bisa mengerti strategi-strategi yang cocok digunakan dalam sebuah permasalahan sehingga algoritma yang dibuat lebih efisien. Terima kasih juga kepada ibu dari penulis yang selalu mendukung dalam doa maupun makanan ketika penulis begadang untuk mengerjakan makalah ini. Juga kepada teman-teman Teknik Informatika ITB angkatan 2019 yang telah memberi penulis inspirasi dan pertolongan sehingga makalah ini dapat selesai.

## Referensi

[1] Dr. Andrew Davison, “Pattern Matching,” WiG Lab (teachers room), CoE, 2006.

[2] Giovanni Organtini, “Regular Expressions,” Linux Journal, 1 Mei 2003. (diakses 11 Mei 2021, 09.51 WIB) Diambil dari: https://www.linuxjournal.com/article/6324

[3] guru99, “Strings in C: How to Declare Variable, Initialize, Print, Example.” (diakses 10 Mei 2021, 12.55 WIB) Diambil dari : https://www.guru99.com/c-strings.html

[4] Ivan Lanin, “Huruf Miring,” in PUEBI Daring. (diakses 11 Mei 2021, 10.50 WIB) Diambil dari: https://puebi.readthedocs.io/en/latest/huruf/huruf-miring/

[5] Masayu Leylia Khodra, “String Matching dengan Regular Expression,” 2019. (diakses 11 Mei 2021, 10.50 WIB) Diambil dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf

[6] Paul E. Black, "string matching", in Dictionary of Algorithms and Data Structures [online], Paul E. Black, ed. 2 November 2020. (diakses 10 Mei 2021, 12.55 WIB) Diambil dari: https://www.nist.gov/dads/HTML/stringMatching.html

## 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.

Cirebon, 10 Mei 2021

Jeremia Axel 1359188