Aplikasi Algoritma String Matching dalam Identifikasi dan Filtrasi Email Spam

Gagas Praharsa Bahar - 13520016
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13520016@std.stei.itb.ac.id

***Abstract*—*Electronic mail* atau e-mail adalah salah satu bentuk komunikasi yang umum digunakan sejak kehadiran Internet. E-mail sudah menjadi pengganti alamat secara digital dan adalah sesuatu yang vital sebagai gambaran manusia secara daring. E-mail dapat juga digunakan oleh orang-orang tidak bertanggungjawab untuk melakukan hal-hal yang tidak baik, seperti contohnya mengirimkan e-mail spam. Oleh karena itu, dibuatlah sebuah algoritma untuk mengidentifikasi e-mail spam dengan memanfaatkan algoritma *string matching*. Algoritma *string matching* dapat membantu mencocokkan isi dari e-mail dengan kata-kata spam yang berada pada *database*.**

***Keywords*—string matching; spam; email; boyer-moore; kmp; bruteforce**

I. PENDAHULUAN

Dewasa ini, perkembangan teknologi membawa banyak perubahan dalam kehidupan kita. Kehadiran teknologi baru ini mengakibatkan adanya perubahan pula dalam cara kita hidup sebagai manusia secara fundamental. Banyak inovasi dari teknologi yang mempermudah kehidupan manusia, dengan salah satu inovasi yang paling revolusioner adalah kehadiran Internet. Kehadiran Internet membuat manusia dapat berbagi informasi dan berkomunikasi dengan manusia lain yang ada di belahan bumi lainnya. Ditambah lagi dengan kehadiran ponsel pintar atau *smartphone* yang sudah seperti kebutuhan primer dalam kehidupan kita, manusia dalam masyarakat modern selalu terhubung dan terkoneksi dengan manusia lainnya, lewat Internet. Sejatinya, Internet hanyalah sebuah jaringan yang menyambungkan perangkat-perangkat yang terkoneksi kepadanya. Untuk benar-benar berkomunikasi dengan orang lain, terdapat banyak pilihan yang dapat dipakai, seperti contohnya email, instant messaging, SMS, VoIP (Voice over Internet Protocol), video conference, dan masih banyak yang lainnya.

Electronic mail atau e-mail adalah sebuah aplikasi berbasis komputer yang diperuntukkan sebagai sebuah sistem pertukaran pesan antar pengguna. Pesan ini dapat berupa teks, grafik, suara, ataupun gambar dengan animasi. Pada umumnya, pesan ini dapat langsung dikirimkan sekaligus kepada banyak pengguna. Para pengguna yang termasuk dalam jaringan sistem email umumnya memiliki kotak pos elektronik bersamaan dengan alamat e-mail yang dapat menerima dan menyimpan pesan hasil korespondensi dari pengguna. Banyak dari sistem e-mail juga memiliki fitur-fitur tambahan seperti menotifikasi pengguna ketika ada pesan masuk sampai mengecek e-mail yang diterima apakah e-mail tersebut termasuk ke dalam e-mail spam atau bukan. E-mail sudah menjadi salah satu sarana komunikasi yang sangat umum di Internet, dan seringkali menjadi syarat untuk mendaftarkan sebuah akun pada situs lain di internet. Umumnya, provider e-mail juga menyediakan layanannya secara gratis, seperti Google, Yahoo, Hotmail, dan lain sebagainya.

Pada penggunaan e-mail, tidak semua pesan adalah pesan yang penting dan bermakna. Terkadang, banyak sekali pesan pesan tidak penting yang berupa promosi ataupun penipuan dalam skala masif. Pesan-pesan seperti ini dikenal juga dengan nama pesan spam. Pesan spam dapat mengganggu produktivitas pengguna e-mail dan dapat menjerumuskan ke dalam tindak kriminal seperti penipuan secara daring. Oleh karenanya, umumnya penyedia e-mail memberikan sebuah sistem filtrasi e-mail spam untuk membantu menyaring pesan-pesan yang kemungkinan termasuk ke dalam kategori spam.

II. TEORI DASAR

A. *Electronic mail (E-mail)*

Electronic mail atau e-mail adalah salah satu sarana komunikasi berbasis komputer yang memanfaatkan keberadaan Internet sebagai perantaranya sebagai sebuah sistem pertukaran pesan antar pengguna. Pesan ini dapat berupa teks, grafik, suara, ataupun gambar dengan animasi. Pengguna memiliki sebuah alamat e-mail (*email address*) yang bertindak sebagai kotak pos elektronik yang dapat menerima dan menyimpan pesan hasil korespondensi dari pengguna.

Pada penggunaannya, terkadang ada juga pesan yang tidak berguna atau sampah, seperti promosi-promosi dan penipuan daring yang biasa terjadi. E-mail seperti ini biasa disebut sebagai spam, dan e-mail spam dapat dideteksi dengan mengecek isi dari e-mail dan mendeteksi kata-kata kunci yang seringkali ada pada e-mail berjenis spam. Penyedia e-mail pada umumnya juga telah memiliki fitur bawaan untuk menyaring spam pada e-mail.

B. *String*

String adalah sebuah sekuens karakter, baik huruf, angka, tanda baca, dan karakter-karakter valid lainnya. String biasanya diimplementasikan sebagai *array of char* yang menyimpan

sekeuns elemen *char* dengan sebuah enkoding karakter. Pada umumnya, string dikelilingi dengan *double quote*, dan diakhiri dengan *null terminator* atau '\0'.

Istilah lainnya yang perlu diketahui tentang *string* antara lain:

1. Prefix: *substring* dari S yang dimulai dari huruf pertama sampai huruf ke-k.
2. Suffix: *substring* dari S yang dimulai dari huruf ke-k sampai huruf terakhir, dengan k adalah index apapun diantara 0 sampai m-1 dengan m adalah panjang string.

C. Algoritma String Matching

String Matching atau pencocokan string adalah sebuah algoritma yang digunakan untuk mencari kemunculan suatu string atau yang selanjutnya disebut *pattern* dalam sebuah teks. *Pattern* adalah sebuah pola yang diharapkan muncul pada teks, sehingga panjang *Pattern* harus lebih pendek dari panjang teks.

Sebagai contoh, pada pengecekan sekuens penyakit pada dna, sekuens penyakit bertindak sebagai pattern dan sekuens dna manusia bertindak sebagai teks. Algoritma *String Matching* akan mengecek apakah sekuens pattern pernah muncul pada teks yang diberikan. Algoritma *String Matching* memiliki beberapa penerapan selain yang telah disebutkan, contohnya pendeteksi plagiarisme, forensik digital, cek ejaan, analisis citra, dan lain-lain.

D. Jenis-jenis Algoritma String Matching

Implementasi algoritma string matching tidak hanya satu, melainkan ada beberapa algoritma berbeda yang memiliki alur berpikir yang berbeda pula. Beberapa algoritma *string matching* yang terkenal antara lain:

1. Algoritma KMP (Knuth-Morris-Pratt)
2. Algoritma Boyer-Moore
3. Algoritma *Naïve* / *Brute Force*

Penjelasan lebih lanjut mengenai ketiga algoritma tersebut akan ada pada poin-poin selanjutnya.

E. Algoritma Brute Force String Matching

String Matching dapat diimplementasikan dengan pendekatan *brute force*. Proses kerja dari algoritma brute force adalah sebagai berikut:

1. Bandingkan karakter pertama dari *pattern* dengan karakter pertama yang ada pada teks.
2. Apabila sama, pengecekan dilanjutkan ke indeks selanjutnya pada *pattern* maupun teks.
3. Apabila berbeda, pengecekan dilanjutkan ke indeks berikutnya pada teks, dan *pattern* diulang dari indeks pertama.
4. Ulangi langkah satu dan dua sampai semua karakter dalam *pattern* telah dicocokkan, apabila sudah, maka terdapat *pattern* pada teks.

Dengan panjang karakter dari *pattern* dilambangkan sebagai m dan panjang karakter dari teks dilambangkan sebagai n maka kompleksitas algoritma *brute force* adalah:

- Worst case: Jumlah perbandingan = m(n-m+1) = O(mn). Contoh:

  Teks: aaaaaaaaaaaaaaaaaaaaaaaaah

  Pattern: aaah

- Best case: terjadi apabila karakter pertama pattern P tidak pernah sama dengan karakter teks T yang dicocokkan. Jumlah perbandingan = n = O(n).

  Teks: abcdefghijklmnopqrstuvwxyzzz

  Pattern: zzz

- Average case: terjadi pada kasus umumnya dengan kompleksitas algoritma O(m+n). Ini adalah kompleksitas algoritma yang tidak buruk.

  Teks: iniadalahcontohabcdstringmatching

  Pattern: abcd

Dikarenakan best case dari brute force terjadi apabila karakter pertama pattern tidak pernah sama dengan teks, maka algoritma ini akan lebih cepat apabila jumlah alfabet yang ada pada teks banyak (contoh: A..Z + a..z + 1..9 …) dan akan menjadi lebih lambat apabila set alfabetnya kecil (contoh: binary, 0 dan 1).

F. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP adalah algoritma yang dibuat oleh James H. Morris yang kemudian ditemukan kembali oleh Donald Knuth tidak lama setelahnya. Morris dan Vaughan Pratt juga mempublikasikan laporan tentang algoritma ini pada 1970. Pada tahun 1977, Knuth, Morris, dan Pratt mempublikasikan algoritma ini bersama-sama.

Cara kerja algoritma ini sama dengan *brute force* dalam hal urutan pencarian yaitu dimulai dari kiri ke kanan, namun penentuan kapan penunjuk indeks yang sedang dicek akan digeser berbeda dan lebih teroptimasi.

Pada algoritma KMP, terdapat fungsi pembantu yang disebut sebagai *border function* b(k), dimana b(k) didefinisikan sebagai ukuran *prefix* terbesar dari P[0..k] yang juga merupakan suffix dari P[1..k]. Fungsi ini dibutuhkan karena apabila ada sebuah ketidakcocokan, jumlah pergeseran maksimal yang dapat dilakukan agar tidak ada perbandingan yang sia-sia adalah sebesar keluaran dari *border function* b(k) tersebut. *Border function* ini sendiri seringkali diterapkan sebagai sebuah *array*.

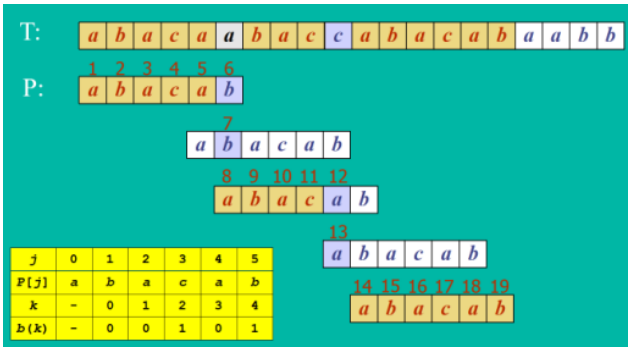


Fig. 1. Contoh penerapan algoritma KMP dan *border function.*

Dengan panjang karakter dari *pattern* dilambangkan sebagai m dan panjang karakter dari teks dilambangkan sebagai n maka kompleksitas algoritma *brute force* adalah:

- Penghitungan border function: O(m)
- Pencarian string: O(n)
- Kompleksitas total algoritma KMP: O(m+n).

Algoritma ini lebih cepat apabila dibandingkan dengan algoritma *brute force*, karena algoritma ini tidak melakukan perbandingan yang sia-sia. Selain itu, algoritma ini juga tidak pernah memundurkan *pointer*, sehingga cocok untuk memproses *file* yang diberikan dalam bentuk *stream*. Namun, kebalikan dengan *brute force*, KMP akan lebih buruk apabila jumlah alfabet bertambah, karena apabila ada ketidakcocokan pada awal karakter, algoritma ini akan menjadi lebih lambat.

G. Algoritma Boyer-Moore

Algoritma Boyer-Moore adalah salah satu algoritma yang cukup efektif dalam *string matching*. Algoritma ini dikembangkan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977.

Algoritma Boyer-Moore memanfaatkan dua teknik yang unik, yaitu:

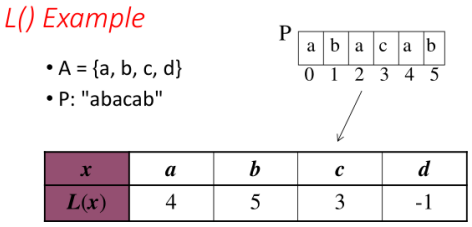
- The *looking-glass* technique

  Teknik ini mencari *pattern* pada teks dengan pola terbalik (dari kanan ke kiri).

- The *character-jump* technique

  Teknik ini berarti indeks akan lompat dengan cara yang berbeda, tergantung dengan kondisi yang ada. Kondisi pertama adalah ketika *mismatch* terjadi pada karakter tertentu (misal x), maka coba untuk menggeser P ke kanan untuk meluruskan kemunculan terakhir x pada P dengan T[i]. Kondisi kedua adalah apabila P memiliki x lagi namun tidak bisa menggeser P ke kanan, maka geser P sebanyak 1 karakter ke T[i+1]. Apabila keduanya tidak terpenuhi, maka geser P sampai P[0] dan T[i+1] lurus.

Dikarenakan banyak digunakan kemunculan terakhir atau *last occurrence* pada algoritma Boyer-Moore, maka dibuat sebuah fungsi *Last Occurrence Function* yang berfungsi untuk memproses pattern P dan alphabet A terlebih dahulu. L(x) terdefinisi sebagai indeks terbesar i yang membuat P[i] == x, atau -1 apabila tidak ada indeks yang memenuhi.



L() stores indexes into P[]

Fig. 2. Contoh last occurrence function.

Pada umumnya, L() dihitung oleh program saat pattern dibaca, dan hasil perhitungannya akan disimpan sebagai *array*, mirip dengan tabel yang ada pada contoh.

Algoritma ini memiliki kompleksitas terburuk O(nm + A). Namun, Boyer-Moore, sama seperti *brute force*, akan lebih baik ketika jumlah huruf pada alphabet besar, dan lambat apabila alphabet kecil, sehingga lebih cocok untuk *matching* pada teks bahasa Inggris, misalnya, dibandingkan dengan sesuatu seperti *binary*.

III. Penerapan String Matching

Pada penerapan algoritma *string matching* pada filtrasi e-mail spam, akan digunakan algoritma *brute force*, KMP, dan *Boyer-Moore* dalam mendeteksi sebuah e-mail, apakah e-mail tersebut termasuk spam atau bukan. Ketiga algoritma akan dibandingkan dan diberikan hasil ujinya untuk mengetahui algoritma mana yang paling cepat dan paling cocok untuk menjadi implementasi dari *string matching* untuk e-mail spam.

A. Mekanisme Pengujian Algoritma String Matching

Untuk menguji, digunakan sebuah program yang ditulis oleh penulis dengan data uji dan kata kunci spam yang didapatkan dari berbagai sumber. Program ditulis dengan bahasa Python dengan spesifikasi algoritma sebagai berikut:

- Dapat menerima masukan berupa teks e-mail.
- Dapat mengakses database yang berisi kata kunci spam.
- Program dapat dimodifikasi dengan mudah untuk menggunakan algoritma jenis lain.
- Program mengembalikan hasil keputusan e-mail spam atau tidak dan waktu yang dibutuhkan.

Sumber dataset adalah Enron-Spam dengan spesifiknya dataset Enron1. Dataset ini terdiri dari 1500 email terduga spam dan 3600 email normal. Karena set kata tidak lengkap, tidak semuanya akan terdeteksi menjadi spam. Pada eksperimen ini, hanya akan dicoba pada 100 email dengan spesifikasi 50 email spam dan 50 email normal untuk dicoba.

Dalam prakteknya, e-mail akan berlaku sebagai teks dan kata-kata terduga spam akan satu persatu dicek sebagai pattern. Untuk percobaan kali ini, digunakan 17 kata spam terbanyak sebagai pattern yang didapatkan dari dataset Enron1.

Contoh e-mail adalah sebagai berikut:

```
Subject: your prescription is ready . . oxwq s f e
low cost prescription medications
soma , ultram , adipex , vicodin many more
prescribed online and shipped
overnight to your door ! !
one of our us licensed physicians will write an
fda approved prescription for you and ship your
order overnight via a us licensed pharmacy direct
to your doorstep . . . . fast and secure ! !
click here !
no thanks , please take me off your list
ogrg z
lqlokeolnq
lnu
```

Contoh pattern adalah sebagai berikut:

```
prescription
viagra
click here
```

B. Implementasi Algoritma String Matching

Dalam implementasi algoritma *string matching* pada penggunaannya untuk mengidentifikasi spam, penulis membuat implementasi tiga algoritma dalam bahasa pemrograman Python, yaitu:

- Algoritma Brute Force

```
def BruteForce(str, substr):
  len_sub = len(substr)
  len_str = len(str)
  if len_sub > len_str:
    return 0
  elif len_sub == len_str:
    if str == substr:
      return 1
    return 0
  i = 0
  for i in range(0, len_str - len_sub + 1):
    j = 0
    while (j < len_sub and str[i+j] == substr[j]):
      j += 1
    if j == len_sub:
      return 1
  return 0
```

- Algoritma KMP

```
def CalculatePrefixSuffix(substr):
        table = [0 for i in range(len(substr) + 1)]
        table[0] = 0
        j = 0
        i = 1
        while (i < len(substr)):
                if substr[i] == substr[j]:
                        table[i] = j + 1
                        i += 1
                        j += 1
                elif j == 0:
                        table[i] = 0
                        i += 1
                else:
                        j = table[j - 1]
        return table

def KMP(str, substr):
        table = CalculatePrefixSuffix(substr)
        len_sub = len(substr)
        len_str = len(str)
        if len_sub > len_str:
                return 0
        elif len_sub == len_str:
                if str == substr:
                        return 1
                return 0
        i, j = 0, 0
        while (i < len_str):
                if str[i] == substr[j]:
                        i += 1
                        j += 1
                elif j == 0:
                        i += 1
                else:
                        j = table[j-1]
                if len_sub == j:
                        return 1
        return 0
```

- Algoritma Boyer-Moore

```
def CalculateTable(substr):
        table = [0 for i in range(257)]
        for i in range(0, 257):
                table[i] = -1
        for i in range(0, len(substr)):
                table[substr[i]] = i
        return table
```

Sebelum e-mail dicek menggunakan algoritma *string matching*, string terlebih dahulu dilakukan cleaning untuk menghapus whitespace yang tidak perlu dan tanda baca yang tidak perlu juga. E-mail juga akan dibuat menjadi lowercase secara menyeluruh agar tidak ada *case* yang luput karena permainan huruf kapital.

C. *Hasil Pengujian*

Setelah menguji ketiga algoritma dengan 100 buah *testcase* yang terdiri dari 50 e-mail *ham* (asli) dan 50 e-mail *spam*, maka didapatkan hasil seperti berikut:

```
def Boyermoore(string, substr):
        table = CalculateTable(substr)
        len_sub = len(substr)
        len_str = len(str)
        if len_sub > len_str:
                return 0
        elif len_sub == len_str:
                if str == substr:
                        return 1
                return 0
        i = 0
        for i in range(-1, len_str - len_sub):
                j = len_sub - 1
                while (j >= 0 and str[i+j] ==
substr[j]):
                        j -= 1
                if j < 0:
                        return 1
                slide = j - table[str[i+j]]
                if slide < 1:
                        slide = 1
                i += slide
        return 0
```

```
Brute Force: 0.10899996757507324 s
Spam count: 7
KMP: 0.12396693229675293 s
Spam count: 7
Boyer-Moore: 0.29488563537597656 s
Spam count: 7
```

Fig. 3. Percobaan pertama pada dataset 100 email.

```
Brute Force: 0.16800403594970703 s
Spam count: 4
KMP: 0.14502882957458496 s
Spam count: 4
Boyer-Moore: 0.35259342193603516 s
Spam count: 4
```

Fig. 4. Percobaan kedua pada dataset 100 email lainnya.

```
Brute Force: 0.07401013374328613 s
Spam count: 7
KMP: 0.07600855827331543 s
Spam count: 7
Boyer-Moore: 0.18573760986328125 s
Spam count: 7
```

Fig. 5. Percobaan ketiga pada dataset 100 email lainnya.

Atau dalam bentuk tabel rekapitulasi ketiga percobaan,

Algoritma	Spam count	Time taken
Brute Force	18	0.351s
KMP	18	0.345s
Boyer-Moore	18	0.831s

Fig. 6. Tabel rekapitulasi ketiga percobaan.

Dapat disimpulkan bahwa ketiga algoritma sudah benar dan mendeteksi jumlah spam yang sama, yaitu sebanyak 18. Untuk kecepatannya sendiri, algoritma KMP menjadi yang tercepat, mengalahkan Brute Force dan BM. Algoritma BM disini kurang teroptimisasi karena harus menghitung tabel yang menyita waktu untuk setiap kata, sehingga waktunya sedikit lebih lambat apabila ukuran test kata lebih panjang. Walaupun demikian, ketiga algoritma terbukti dapat mendeteksi spam dengan baik apabila diberikan susunan kata penghasil spam yang cukup baik. Algoritma masih dapat dioptimisasi lebih lanjut sesuai dengan kebutuhan implementasi spesifik dari domain.

IV. Kesimpulan dan Saran

Algoritma *String Matching* adalah satu dari sekian banyak algoritma yang dapat diterapkan pada berbagai jenis masalah di dunia nyata. Salah satu kegunaan *String Matching* adalah dalam mengidentifikasi apakah sebuah pesan elektronik atau *e-mail* termasuk ke dalam kategori spam atau tidak penting. Pada pengaplikasian ini, algoritma *String Matching* membantu untuk mencocokkan kata-kata yang terdapat pada *e-mail* (yang berlaku sebagai teks) dengan kata-kata yang terdapat pada database sebagai kata yang dicurigai merupakan spam. Aplikasi ini hanyalah satu dari beragam aplikasi lainnya dari algoritma *string matching*. Untuk kedepannya, apabila ingin melanjutkan penelitian yang ada pada makalah ini, dapat dicoba untuk diimplementasikan dengan menggunakan *machine learning* agar dapat terbentuk model yang terlatih untuk mendeteksi spam.

Link Video Youtube

Untuk mempermudah konsumsi artikel dan lebih menyebarluaskan hasil temuan dari makalah ini, penulis juga mencantumkan pranala video YouTube untuk ditonton: https://youtu.be/2uK5Y68pdLI

UCAPAN TERIMA KASIH

Penulis mengucapkan puji dan syukur kepada Tuhan Yang Maha Esa atas berkah dan rahmatnya sehingga makalah ini yang berjudul “Aplikasi String Matching dalam Identifikasi dan Filtrasi Email Spam” dapat diselesaikan dengan baik dan lancar. Penulis juga mengucapkan terima kasih kepada seluruh dosen mata kuliah IF2211 Strategi Algoritma tahun akademik 2021/2022 yang sudah membagikan ilmunya kepada seluruh mahasiswa Teknik Informatika ITB angkatan 2020. Ucapan terima kasih tak lupa penulis ucapkan kepada orangtua dan keluarga yang selalu mendukung penulis, dan teman-teman terdekat penulis yang memberikan dukungan baik dari segi mental maupun materil.

DAFTAR PUSTAKA

[1] https://www.britannica.com/technology/e-mail. Diakses pada 20 Mei 2022.

[2] https://press.rebus.community/programmingfundamentals/chapter/string-data-type/. Diakses pada 20 Mei 2022.

[3] I Knuth, Donald; Morris, James H.; Pratt, Vaughan (1977). "Fast pattern matching in strings". SIAM Journal on Computing. 6 (2): 323–350.

[4] Boyer, Robert S.; Moore, J Strother (October 1977). "A Fast String Searching Algorithm". Comm. ACM. New York: Association for Computing Machinery. 20 (10): 762–772.

[5] Munir, Rinaldi. 2021. Pencocokan string (string matching). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf. Diakses pada 20 Mei 2022.

[6] http://nlp.cs.aueb.gr/software_and_datasets/Enron-Spam/index.html. Diakses pada 21 Mei 2022.

[7] Asif, Karim. 2019. https://www.researchgate.net/figure/A-sample-of-spam-keywords-and-their-frequency-in-the-Enron-spam-dataset_tbl2_333737669. Diakses pada 21 Mei 2022.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2022

Gagas Praharsa Bahar
13520016