Penerapan Algoritma String Matching untuk Filterisasi Inappropriate Comment

Arief Darmawan Tantriady 13518015
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : ariefdarmawantantriady@gmail.com

***Abstraksi*—Seiring perkembangan zaman, interaksi antar manusia menjadi lebih mudah, tanpa terbatasi jarak,usia,bidang pekerjaan dan lainnya. Tidak dipungkiri juga, seiring waktu kemudahan untuk berpendapat berkembang terus menerus. Terutama, dengan adanya sosial media, memungkinkan siapapun dan dimanapun dapat mengemukakan pendapatnya. Hal ini cukup baik bila dimanfaatkan secara positif, namun tentu saja bisa memicu hal-hal negatif seperti komentar kurang pantas yang sering ditemui pada akun-akun atau post sosial media. Makalah ini membahas salah satu cara pembatasan hal-hal negatif tersebut dengan menyaring komentar yang layak dengan memanfaatkan algoritma *String Matching*.**

Keywords*— *String Matching, sosial media

I. Pendahuluan

Seperti yang kita tahu, penggunaan internet banyak macamnya salah satunya sosial media, sosial media ibarat koin bersisi dua dalam hal dampaknya ke kehidupan manusia. Di satu sisi, sosial media memudahkan orang-orang untuk saling berinteraksi, tanpa terbatasi jarak. Sosial media juga memudahkan orang-orang untuk menyuarakan pendapatnya. Di sisi lain, dengan dimudahkannya orang-orang untuk berinteraksi satu sama lain, justru memudahkan orang-orang untuk saling bertikai melalui kata-kata, *cyber-bullying, hate speech*, dan lain-lain. Tak jarang juga, orang memanfaatkan kebebasan berpendapat di dunia maya untuk menggiring opini yang menjatuhkan satu sisi atau menebar kebencian. Tujuan dibuatnya makalah ini yaitu menawarkan sebuah solusi untuk menyaring comment yang mengandung isi yang kurang pantas. Cara yang ditawarkan yaitu dengan algoritma *String Matching* untuk menyaring kata-kata yang biasanya dipakai dalam kalimat yang kurang pantas. Algoritma ini bekerja dengan mengidentifikasi sebuah kalimat mengandung kata-kata yang biasanya dipakai pada comment yang kurang pantas.

II. Dasar Teori

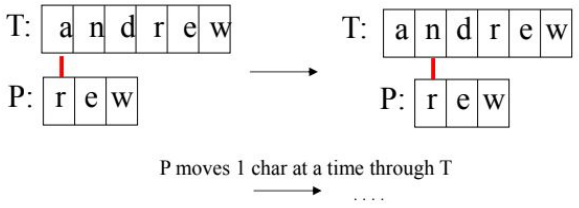
String Matching adalah algoritma pencocokan *pattern* dengan *text*. Text adalah kalimat/paragraf/teks yang panjangnya n karakter. Pattern adalah *string* dengan panjang m karakter yang akan dicari di dalam teks.
Contoh :

T: the rain in spain stays mainly on the plain
P : main
Hasil : the rain in spain stays mainly on the plain
Konsep pada *String:*
Prefix : String yang berupa substring dari S[0..k]
Suffix : String yang berupa substring dari S[k..m-1]
Contoh:
Misalnya , S = “andrew”
Prefix : “a”,”an”,”and”,”andr”,”andrew”
Suffix : “w”,”ew”,”rew”,”drew”,”ndrew”,”andrew”

Ada 4 jenis algoritma yang biasanya digunakan untuk *String Matching*,yaitu:

A. *String Matching dengan Bruteforce*

Algoritma ini mengecek setiap posisi dalam text T apakah pattern P mulai pada posisi itu.



Gambar 2.1 Contoh Algoritma String Matching dengan Bruteforce
sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

```
Teks: NOBODY NOTICED HIM
Pattern: NOT

  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

Gambar 2.2. Contoh Algoritma String Matching dengan Bruteforce
sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

Algoritma Bruteforce dalam bahasa Java:

```
public static int brute(String text,String pattern)
{
        int n = text.length(); // n is length of text
        int m = pattern.length(); // m is length of
pattern
        int j;
        for(int i=0; i <= (n-m); i++) {
        j = 0;
        while ((j < m) && (text.charAt(i+j)==
pattern.charAt(j))){
        j++;
        }
        if (j == m)
        return i; // match at i
        }
        return -1; // no match
} // end of brute()
```

Kompleksitas:

- **Worst Case**
  Jumlah perbandingan : m(n – m + 1) = O(mn).
  Contoh :
  T: aaaaaaaaaaaaaaaaaaaaaaaaaah
  P: aaah

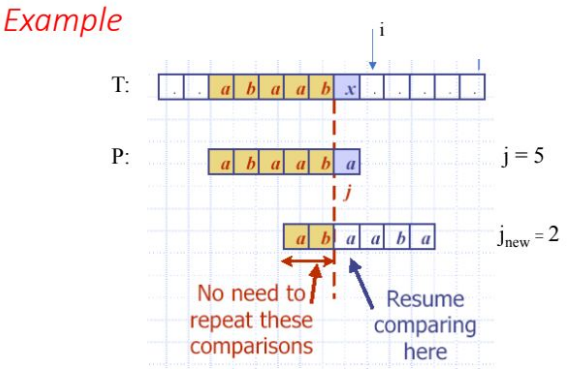
- **Average Case**
  Jumlah perbandingan : m+n = O(m+n)
  Contoh :
  T: a string searching example is standard
  P: store

- **Best Case**
  Terjadi saat karakter pertama *pattern* tidak pernah sama dengan karakter teks T yang dicocokkan.
  Jumlah perbandingan : maksimal n kali.
  Contoh :
  T: String ini berakhir dengan zzz
  P: zzz

Algoritma *Bruteforce* cepat kalau alfabet yang digunakan pada teks cukup luas. Tetapi, algoritma *bruteforce* lambat saat alfabet yang digunakan sempit (misalnya : 0,1).

B. Knuth-Morris Pratt Algorithm

Algoritma ini mencari *pattern* di *text* dengan *scanning* kiri ke kanan (seperti *Bruteforce)*. Yang membedakan antara *bruteforce* dengan algoritma ini adalah dalam proses penggeserannya.
Jika ketidakcocokan terjadi antara *text* dan *pattern* pada P[j], maka besar pergeseran yang dapat dilakukan kepada *pattern* untuk menghindari pencocokan yang kurang efisien yaitu: *prefix* terpanjang dari P[0..j-1] yang juga merupakan *suffix* dari P[1..j-1]



Gambar 2.3. Contoh Algoritma String Matching dengan KMP
sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

jnew=2 karena:

- *Prefix* terpanjang dari *pattern*:
  abaab (P[0..4])
  yang merupakan suffix dari *pattern*:
  abaab
- Ditemui : ab (panjang = 2)
- Jumlah pergeseran :
  s = panjang(*pattern)* - panjang (ab)
  = 5-2 = 3

Untuk menghitung jnew pada setiap *index pattern*, sebelum dilakukan pencocokan *pattern* dengan *text*, *pattern* dilewati terlebih dahulu ke tahap *preprocessing* yaitu tahap penghitungan fungsi pinggiran (*border function*).

Fungsi Pinggiran (*Border Function)*:
Border function b(k) adalah ukuran terpanjang dari *prefix* P[0..k] yang juga merupakan *suffix* dari P[1..k].
dimana: k adalah posisi sebelum ketidakcocokan (j-1)
j adalah posisi ketidakcocokan

Border Function Example

(k = j-1)

➢P: abaaba
j: 012345

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.

➢In code, b() is represented by an array, like the table.

• Contoh lain: P = abababababca
j = 0123456789

(k = j-1)

j	0	1	2	3	4	5	6	7	8	9
P [j]	a	b	a	b	a	b	a	b	c	a
k	-	0	1	2	3	4	5	6	7	8
b[k]	-	0	0	1	2	3	4	5	6	0

Gambar 2.4 Contoh perhitungan Border Function
sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

Tabel yang dihasilkan dari perhitungan *border function* ini nantinya akan digunakan setiap adanya ketidakcocokan antara *pattern* dengan *teks*. Pada algoritma *bruteforce*, jika ada ketidakcocokan maka *pattern* akan dicek dari awal dan indeks dari teks di +1.
Pada algoritma ini, setiap adanya ketidakcocokan (misal pada P[j]) maka *pattern* akan digeser menjadi indeks ke b[k] (j=b(k)) dengan k = j-1.
Algoritma KMP pada bahasa Java:

Border Function

```
public static int[] computeFail(String pattern)
{
        int fail[] = new int[pattern.length()];
        fail[0] = 0;
        int m = pattern.length();
        int j = 0;
        int i = 1;
        while (i < m) {
        if (pattern.charAt(j) == pattern.charAt(i)) { //j+1
chars match
        fail[i] = j + 1;
        i++;
        j++;
        }
        else if (j > 0) // j follows matching prefix
        j = fail[j-1];
        else { // no match
        fail[i] = 0;
        i++;
        }
        }
        return fail;
} // end of computeFail()
```

Matching Function

```
public static int kmpMatch(String text,String pattern)
{
        int n = text.length();
        int m = pattern.length();
        int fail[] = computeFail(pattern);
        int i=0;
        int j=0;
        while (i < n) {
        if (pattern.charAt(j) == text.charAt(i)) {
        if (j == m - 1)
        return i - m + 1; // match
        i++;
        j++;
        }
        else if (j > 0)
        j = fail[j-1];
        else
        i++;
        }
        return -1; // no match
} // end of kmpMatch()
```

Kompleksitas:

Border function: O(m)
String Matching: O(n)
Kompleksitas total: O(m+n)

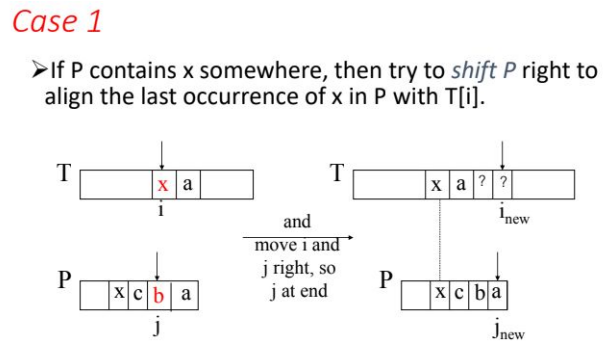
Kelebihan Algoritma KMP:
Tidak pernah mencari mundur dari *text*, algoritma ini bagus untuk pemrosesan *file* yang sangat besar.
Kekurangan Algoritma KMP:
Tidak terlalu bagus saat ukuran dari keragaman alfabet meningkat, akan terjadi banyak sekali ketidakcocokan.

C. Boyer-Moore Algorithm

Algoritma ini memanfaatkan dua teknik:

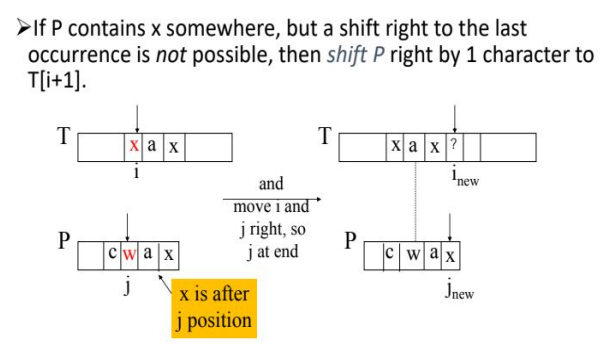
1. *The looking-glass technique*
   Mencari kecocokan *pattern* dengan *text* dimulai dari indeks terakhir *pattern*
2. *The character-jump technique*
   Saat ketidakcocokan terjadi pada T[i]!=P[j], maka ada 3 kemungkinan:

1) P mengandung T[i] dan T[i] tersebut dikiri dari P[j] sekarang.



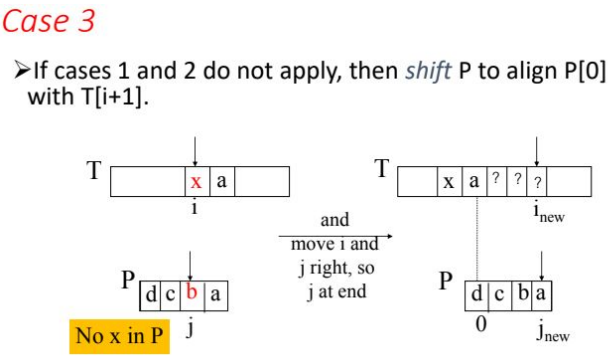
Gambar 2.5. Kemungkinan 1 pada character-jump technique
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

2) P mengandung T[i] namun T[i] tersebut tidak berada dikiri dari P[j] sekarang.



Gambar 2.6. Kemungkinan 2 pada character-jump technique
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

3) P tidak mengandung T[i].

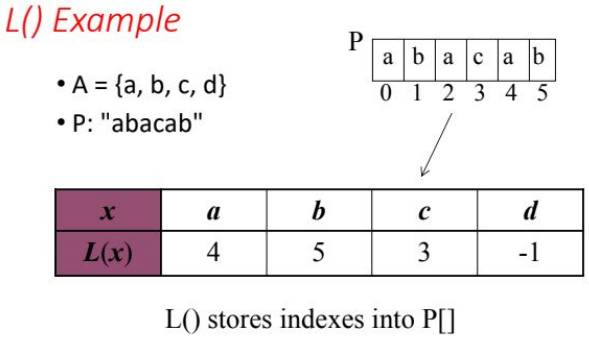


Gambar 2.7.Kemungkinan 3 pada character-jump technique
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

Untuk membantu mempercepat pengecekan apakah *pattern* P mengandung T[i] atau tidak, maka akan dihitung terlebih dahulu *last occurence* dari masing-masing karakter pada P.

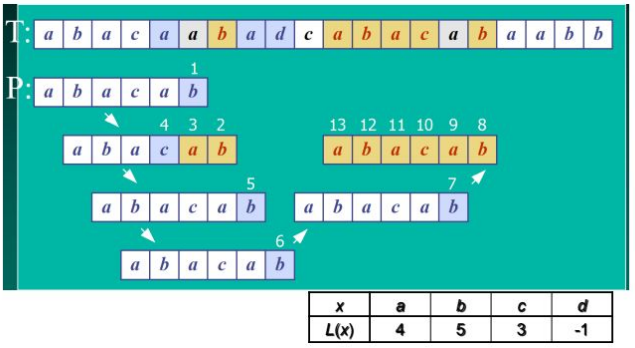
***Last occurrence function*:**
Preprocessing pada algoritma *Boyer-Moore* yang menghitung indeks ditemuinya setiap karakter pada P.
L(x):

- indeks terbesar dari P dimana P[i] = x
- -1 jika tidak ada karakter x



Gambar 2.7. Perhitungan Last Occurence
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

L() ini dihitung saat pattern telah diinput pertama kali, dan disimpan sebagai array berukuran 128.



Gambar 2.8. Contoh String Matching Boyer-Moore
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf

Algoritma *Boyer-Moore* dalam Java:
Last occurrence function:

```
public static int[] buildLast(String pattern)
/* Return array storing index of last
occurrence of each ASCII char in pattern. */
{
        int last[] = new int[128]; // ASCII char set
```

```
        for(int i=0; i < 128; i++)
        last[i] = -1; // initialize array
        for (int i = 0; i < pattern.length(); i++)
        last[pattern.charAt(i)] = i;
        return last;
} // end of buildLast()
```

String Matching function:

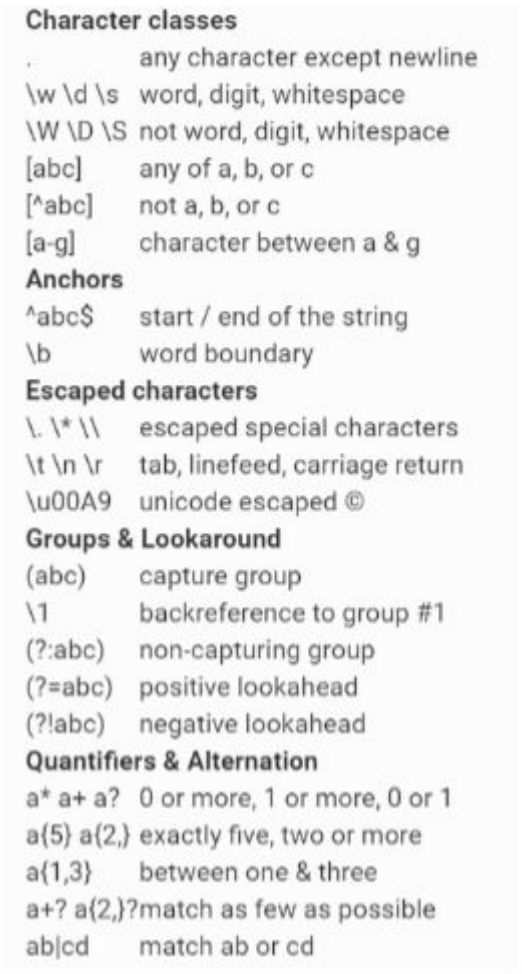
```
public static int bmMatch(String text,String pattern)
{
   int last[] = buildLast(pattern);
   int n = text.length();
   int m = pattern.length();
   int i = m-1;
   if (i > n-1)
          return -1;
   int j = m-1;
   do {
          if (pattern.charAt(j) == text.charAt(i)){
                 if (j == 0)
                        return i; // match
                 else { // looking-glass technique
                        i--;
                        j--;
                 }
          }
          else { // character jump technique
                 int lo = last[text.charAt(i)]; //last occ
                 i = i + m - Math.min(j, 1+lo);
                 j = m - 1;
          }
   } while (i <= n-1);
   return -1; // no match
} // end of bmMatch()
```

Kompleksitas:
Worst case : O(nm+A)
Algoritma ini efektif digunakan saat keragaman alfabet (A) tinggi, dan lambat jika keragaman alfabet (A) rendah.

D. String Matching dengan 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 sudah menjadi standar yang tersebar di semua *tools* dan bahasa pemrograman sehingga penting untuk dipelajari.

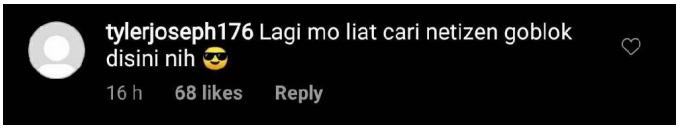
Notasi umum Regex:



Gambar 2.9. Notasi Dasar Regex
Sumber:https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf

III. Langkah Filterisasi Comment

Untuk melakukan penyaringan kata-kata yang kurang pantas, tentu saja sebelumnya diperlukan *database* berupa kata-kata yang biasanya digunakan dalam kalimat yang kurang pantas:



Gambar 3.1. Contoh Komentar Kurang Pantas
Sumber: instagram

Pada gambar diatas, terlihat bahwa komentar tersebut tidak pantas karena mengandung kata "goblok". Jadi kata tersebut akan dimasukkan kedalam *database* kata kurang pantas.

Gambar 3.2. Contoh Komentar Kurang Layak
Sumber:youtube

Pada gambar diatas, terlihat bahwa komentar tersebut sangat tidak pantas karena mengandung kata "cacad", "goblok", "idiot". Semua kata tersebut akan dimasukkan kedalam *database* kata kurang pantas.

Gambar 3.3. Contoh Komentar Kurang Layak
Sumber:instagram

Pada gambar diatas, terlihat bahwa komentar tersebut sopan, namun kurang pantas karena mempromosikan produk pada tempat yang tidak semestinya. Kata-kata berupa "keju mozarella" "keju mozarella khas malang" dapat ditambahkan kedalam *database*.

Setelah menambahkan kata-kata ke dalam *database*. Selanjutnya, tinggal menerapkan *String Matching* algorithm dengan *pattern* berupa setiap kata yang ada di dalam database. Jika terdeteksi adanya *pattern* yang *match* dengan *comment* yang ada. Maka, *comment* tersebut terdeteksi merupakan *comment* yang kurang pantas.

IV. Penerapan Algoritma dalam Bahasa Java

Misalkan , telah ada List<String> inappropriate yang telah diisi dengan kata-kata yang ingin di filter , dan sebuah String *comment* yang berisi *comment* yang ingin dipublikasi Maka, penerapan dari algoritma filter *inappropriate comment* dalam bahasa java:

```
public boolean FilterInappropriate(String
comment,List<String> inappropriate)
{
   for(String filter:inappropriate){

if(bmMatch(comment,filter)!=-1){//memanfaatkan
Boyer-Moore algorithm bisa juga kmpMatch,brute, dan
regex
  //!=-1 berarti salah satu kata kurang pantas terkandung
dalam comment
            return 0;//tidak boleh dipublish
          }
   }
   return 1; // boleh di publish
}
```

Ketepatan fungsi ini bergantung pada List of String inappropriate yang bebas didefinisikan sesuai kebutuhan.

List tersebut mudah dimodifikasi hanya dengan :

- inappropriate.add("Masukkan sebuah string")

  Untuk menambahkan kata yang ingin difilter
- inappropriate.remove("Masukkan sebuah string")

  Untuk menghilangkan kata dari filter

Diutamakan string yang ditambahkan merupakan string yang panjang bukan yang dipisah-pisah agar filter bekerja lebih tepat. Misalnya,"keju mozarella khas malang", bila dipecah jadi "keju","mozarella","khas","malang" jika ada yang berkomentar kata-kata yang mengandung "malang" saja seperti: "Iya itu seperti yang di Malang" maka akan tersaring. Sedangkan bila kita menambahkannya sebagai satu kesatuan "keju mozarella khas malang" maka *comment* tersebut tidak akan tersaring.

V. Kesimpulan

Algoritma *string matching* dapat digunakan untuk menyelesaikan berbagai persoalan, apalagi saat ini, mempublikasi tulisan-tulisan sangat mudah terutama di dunia maya. Pada makalah ini, diberi contoh pemanfaatan *string matching* dalam ranah media sosial terkait *filter comment*.

Video Link di Youtube

Video telah diupload di youtube dengan link:
https://youtu.be/A_PmTBNdAok

Ucapan Terima Kasih

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa karena atas rahmatnya telah memberikan penulis kesempatan untuk menyelesaikan makalah ini. Penulis juga mengucapkan terima kasih kepada semua pihak yang telah membantu dalam pengerjaan makalah ini, khususnya pak Rinaldi Munir selaku dosen penulis pada Mata Kuliah Strategi Algoritma yang telah memotivasi dan memberikan materi-materi terkait Strategi Algoritma sehingga makalah ini dapat terselesaikan.

Referensi

[1] Munir, Rinaldi. 2018. Diakses dari *https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2017-2018/Pencocokan-String-(2018).pdf* pada Sabtu,3 Mei 2020 pukul 10.50 WIB.

[2] Munir, Rinaldi. 2018. Diakses dari *https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf* pada Sabtu,3 Mei 2020 pukul 11.30 WIB.

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, 29 April 2020

Arief Darmawan Tantriady 13518015