# *Hide* Kata Toxic di *Game Online* dengan Algoritma *Knuth-Morris-Pratt*(KMP)

Roy H Simbolon/13519068
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519068@std.stei,itb.ac.id

***Abstract*—Di abad ke-20 ini, era teknologi, *game online* telah berkembang sangat pesat. *Game online* telah menjadi aktivitas favorit banyak orang di waktu senggang, khususnya kaum laki-laki millenial. Aktivitas waktu senggang remaja laki-laki telah beralih dari olahraga ke *game online*, bahkan sampai dipertandingkan dalam suatu turnamen berskala kecil maupun besar. Saat bermain *game online*, ada budaya buruk yang timbul, salah satunya adalah mengeluarkan kata-kata *toxic*. Dan telah banyak *game online* telah melakukan *hide* terhadap kata-kata *toxic* tersebut karena tidak sesuai dengan norma kesopanan. Di dalam makalah ini akan didiskusikan cara mengimplementasikan melakukan *hide* kata-kata *toxic* pada *game online* dengan menggunakan algoritma Knuth-Morris-Pratt.**

***Keywords—Knuth-Morris-Pratt, Game Online, Toxic***

## I. PENDAHULUAN

Di dunia modern, hampir semua orang yang kita temui memiliki akses ke internet. Dengan akses ke internet yang cukup mudah, *game online* telah menjadi aktivitas waktu senggang yang sangat cepat berkembang diminati banyak orang, bahkan banyak orang yang menghabiskan waktu tidak senggangnya bermain *game online*. *Game online* yang bersifat daring membuat penggunanya tidak takut untuk mengeluarkan kata-kata *toxic* karena hanya sekedar adu mulut tanpa adanya main fisik, ditambah tidak ada adanya pengawalan akan etika pengguna seperti yang dialami orang dalam sebuah keluarga, dimana orang yang lebih tua akan marah jika seseorang tidak berlaku sopan akan perkataannya.

Media figure lainnya juga mempengaruhi seseorang dalam keberanian mengeluarkan kata-kata *toxic* dalam bermain *game online*. Seperti *youtube, tiktok,* dan figure lainnya yang menampilkan cara seseorang bermain game dan budaya bermain game yang lagi *trend* dilakukan banyak orang, salah satunya mengeluarkan kata-kata *toxic.*

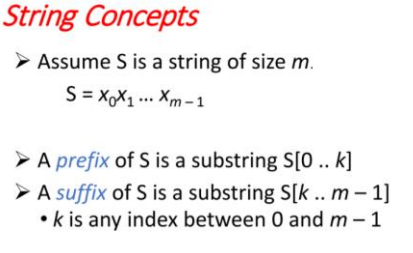
Budaya *toxic* di dalam *game online* membuat semua cemas karena banyak anak-anak kecil yang mengikutinya dan membuat pribadi si anak menjadi pribadi yang lebih liar, karen anak-anak memiliki penangkapan yang lebih cepat akan yang diamatinya daripada orang pada umumnya karena pengetahuan anak-anak masih minim akan yang baik atau salah dan kemampuan otak anak-anak yang masih segar. Oleh karena itu, banyak pula *game online* yang melakukan pencegahan akan hal itu, seperti himbauan waktu batas maksimal bermain *game online*, persetujuan umur, dan melakukan *hide* pada kata-kata yang


dianggap sistem adalah kata-kata *toxic*. Pada umumnya, *game online* melakukan *hide* dengan '*' pada kata-kata *toxic* tersebut agar pengguna lain tidak mengetahui maksud pesan yang mengandung kata tersebut.

Di dalam makalah ini, penulis membahas cara hide kata-kata *toxic* dalam sebuah pesan(string) pada *game online* dengan Algoritma KMP yang diimplementasikan dalam bahasa pemrograman python.

## II. LANDASAN TEORI

Algoritma *string matching* (pencocokan string) adalah algoritma untuk melakukan pencarian semua kemunculan string pendek (pattern) yang muncul dalam teks. Pattern yaitu string dengan panjang m karakter ( m < n ). Teks (text) yaitu *long* string yang panjangnya n karakter. Pada perkuliahan IF2211 2020/2021, ada 3 algoritma *string matching* yang telah diajarkan, yaitu Brute Force, Knuth-Morris-Pratt, dan Boyer-Moore.



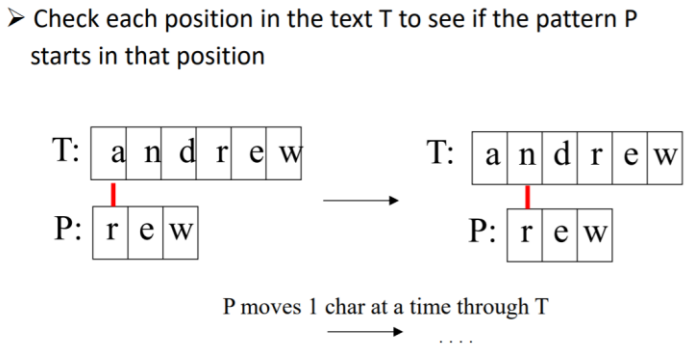

Gambar 2.1 Konsep sebuah string

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

Berikut penjelasan 3 algoritma *string matching* tersebut:

1. Algoritma Brute Force

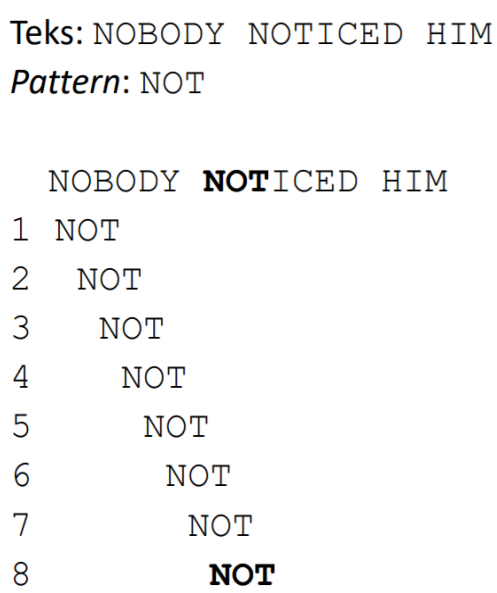
Algoritma brute force merupakan algoritma pencocokan string yang ditulis tanpa memikirkan peningkatan performa. Algoritma ini sangat jarang dipakai dalam praktik, namun berguna dalam studi pembanding dan studi-studi lainnya.Algoritma brute force akan menghitung sebanyak C(n, 2) = n(n – 1)/2 pasangan titik dan memilih pasangan titik yang mempunyai jarak terkecil. Metode brute force dapat digunakan untuk memecahkan hampir sebagian besar masalah.




Gambar2.2 Pengecekan pattern terhadap text pada algoritma Brute Force

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

Pada Algoritma Brute Force, pengecekan dilakukan dengan mengecek untuk setiap karakter T(text) apakah terdapat bagian text yang sama dengan P(pattern) dimulai dari posisi kiri/awal P(pattern) dan dilakukan dengan mengecek per *character*. Saat terjadi *mismatch* atau ketidaksamaan antar *character* pada T(text) dan P(pattern) maka akan dilakukan pergeseran untuk *character* setelahnya pada T(text) untuk di cek lagi. Kompleksitas algoritma ini adalah O(m x n) dimana m adalah panjang teks dan n adalah panjang pola.




Gambar2.3 Contoh Pengecekan pattern terhadap text pada algoritma Brute Force

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

Berikut potongan *source code* implementasi algoritma Brute Force pada java.

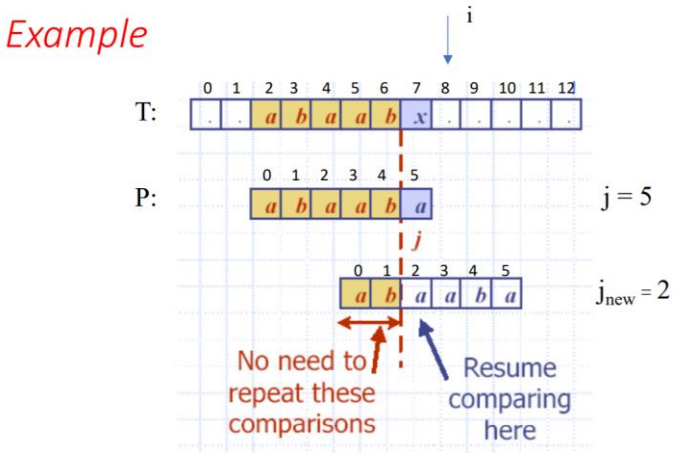
```
public static int brute(String text,String
pattern){
   int n = text.length(); // n is length of
text
   int m = pattern.length(); // m is length of
pattern
   int j;
   for(int i=0; i <= (n-m); i++) {
       j = 0;
       while ((j < m) && (text.charAt(i+j)==
pattern.charAt(j))){
           j++;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

Diambil dari

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

2. Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris Pratt merupakan algoritma pencocokan string yang melakukan pengecekan pattern dari *character* awal hingga *character* akhir pada text layaknya algoritma Brute Force. Namun, pergeseran *character* untuk pengecekan *character* selanjutnya ketika terjadi *mismatch* pada algoritma ini berbeda dengan algoritma Brute Force dan lebih efisien dari algoritma Brute Force.




Gambar2.4 Contoh pergeseran pattern terhadap posisi text pada algoritma Knuth-Morris-Pratt

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

Pada kasus ketika P[j] tidak sama dengan T[i] pergeseran tidak lagi dilakukan satu kali, melainkan dengan melihat nilai panjang prefix pada Pola yang sama dengan postfix pada Pola. Nilai ini disimpan pada sebuah tabel dan dihitung dengan fungsi yang dinamakan *border function*. Nilai pada tabel tersebut menentukan sejauh mana pergeseran yang dilakukan ketika terjadi *mismatch*.

## 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.

Gambar2.5 Contoh tabel *border function* pada algoritma Knuth-Morris-Pratt

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

i adalah indeks *character* pada pattern dan k adalah indeks tabel *border function*, dimana nilai k didapat dari k = j-1 karena saat j=0, belum ada prefix dan suffixnya. Nilai tabel *border function* didapat dari seberapa banyak prefix sama dengan suffix pada indeks tersebut.

## Why is b(4) == 2?

P: "abaaba"

- *b*(4) means
  - find the size of the largest prefix of P[0..4] that is also a suffix of P[1..4]
  - find the size largest prefix of "abaab" that is also a suffix of "baab"
  - find the size of "ab"

  == 2

Gambar2.6 Contoh pencarian nilai tabel *border function* pada algoritma Knuth-Morris-Pratt

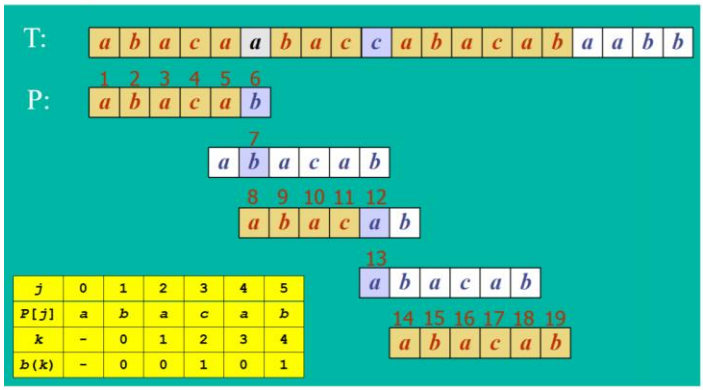
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf

Pada gambar diatas, terdapat contoh pencarian nilai tabel *border function*. b(4)==2? karena *character* b(4) *character* 'b'. dan prefix mempunyai kesamaan dengan suffix sebanyak 2 *character* yaitu 'ab' pada pattern *ab*a*ab* saat kondisi pengecekan b(4).

Dengan adanya tabel *border function*, pengecekan pattern pada text pada algoritma Algoritma Knuth-Morris-Pratt adalah dengan mengecek pattern pada text dimulai dari kiri text dan pengecekan *character* dimulai dari ujung kiri pattern. Ketika terjadi *mismatch* atau ketidakcocokan antar *character* pada text dan pattern, maka dilakukan pergeseran sejauh nilai tabel *border function* indeks *mismatch* - 1 pada pattern. Dan ketika terjadi *mismatch*, maka pengecekan *character* pada pattern terhadap text dimulai dari ujung kiri pattern lagi.

Iterasi adalah banyaknya dilakukan pengecekan *character* sepanjang proses *string match* suatu pattern terhadap sebuah text. Berikut contoh cara menghitung banyaknya iterasi pada sebuah *string match* suatu pattern terhadap sebuah text dengan algoritma Algoritma Knuth-Morris-Pratt.

## Example




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

Gambar2.7 Contoh menghitung iterasi pada *string match* dengan algoritma Knuth-Morris-Pratt

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

Berikut potongan *source code* implementasi algoritma Knuth-Morris-Pratt pada java.

```
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()


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()
```

Diambil dari

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

3. Algoritma Boyer-Moore

Algoritma Boyer Moore merupakan algoritma yang paling mangkus untuk pencocokan string. Tidak seperti algoritma pencarian string yang ditemukan sebelumnya, algoritma Boyer-Moore mulai mencocokkan karakter dari sebelah kanan pattern. Ide di balik algoritma ini adalah bahwa dengan memulai pencocokan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat.Algoritma boyer moore dilakukan dengan dua teknik,yaitu

- *looking-glass technique*

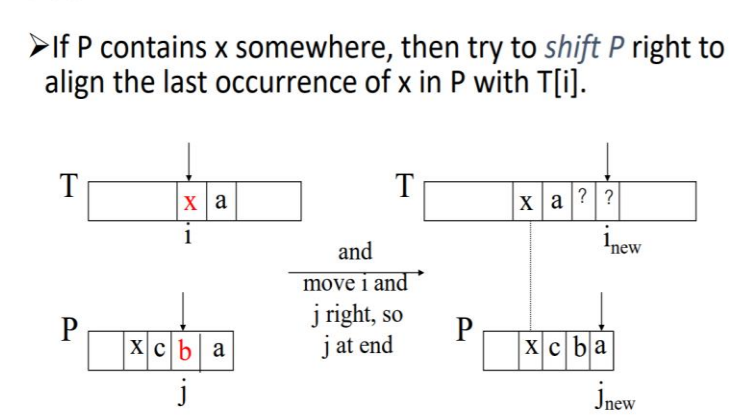
  merupakan teknik mencari pattern pada text dimulai dari bagian ujung akhir pattern.

- *character jump technique*

  merupakan teknik yang dilakukan ketika terjadi *mismatch* text(i) dengan x dan karakter text(i) tidak sama dengan pattern(j). x adalah suatu karakter.

Pada *character jump*. terdapat 3 macam kasus, yaitu:

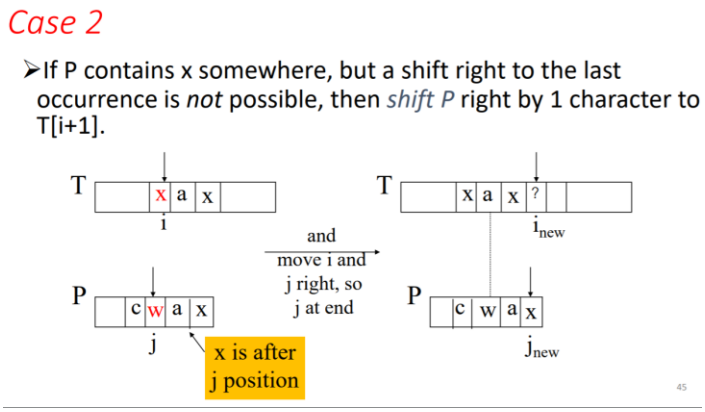
1. Kasus pertama ketika x terdapat di P kemudian dilakukan pergeseran kekanan untuk mensejajarkan x terakhir kali ditemukan pada P




Gambar2.8 Case 1 character jump

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

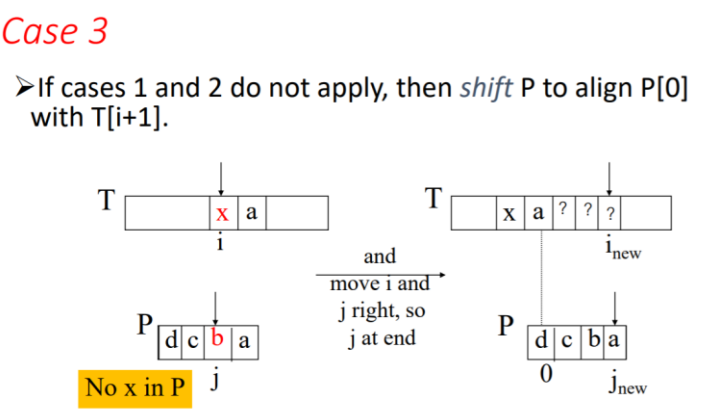
2. Kasus kedua ketika x terdapat di P namun tidak memungkinkan untuk melakukan pergeseran kekanan sehingga P digeser kekanan satu langkah pada T.




Gambar2.9 Case 2 character jump

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

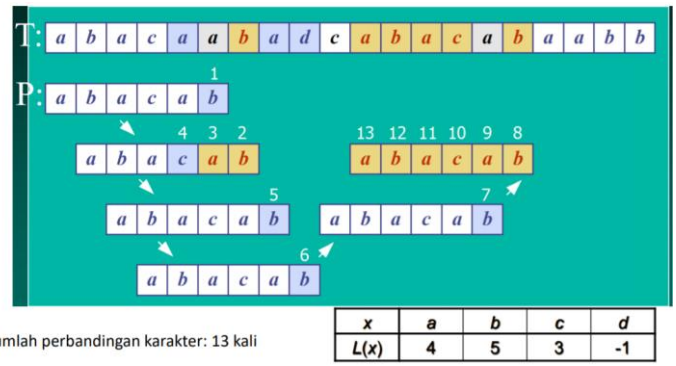
Kasus ketika yaitu ketika kasus pertama dan kedua tidak bisa




lakukan maka sejajarkan P[0] dengan T[i+1]

Gambar2.10 Case 3 character jump https:/Pencocokan-string-2021.pdf

Sama seperti algoritma Brute Force dan algoritma algoritma Knuth-Morris-Pratt, pengecekan pattern dilakukan dengan mengecek per *character*. Namun ada yang beda, dimana pengecekan dilakukan dari *character* yang berada di ujung kanan pattern, ketika ada *mismatch*, *character* yang *mismatch* pada text di cek apakah ada yang sama di kiri *character* pattern yang *mismatch* tadi, jika ada maka disejajarkan, jika tidak pattern digeser ke sebelah kanan sejauh 1 satuan.Berikut contoh menghitung iterasi pada *string matching* dengan algoritma Boyer-Moore.




| x | a | b | c | d |
|---|---|---|---|---|
| L(x) | 4 | 5 | 3 | -1 |

Gambar2.11 Contoh menghitung iterasi pada *string match* dengan algoritma Boyer-Moore

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

Berikut potongan *source code* implementasi algoritma Boyer-Moore pada java.

```
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; // no match if pattern is
                 // longer than text
   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)];
         i = i + m - Math.min(j, 1+lo);
         j = m - 1;
      }
  } while (i <= n-1);
  return -1; // no match
} // end of bmMatch()


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()
```

Diambil dari

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

## III. PEMBAHASAN

Berikut implementasi hide kata-kata *toxic* dalam *game online* dengan menggunakan bahasa pemrograman Python.

Seperti yang telah dibahas pada bagian sebelumnya, pengecekan menggunakan algoritma KMP memerlukan tabel *border function* sebagai acuan besar pergeseran jika terjadi *mismatch* antara *character* yang sejajar/di-cek. Pada implementasi ini, tabel *border function* diperoleh dari *function* `def table_KMP_query(query)`. Kemudian *function* `def algoritma_KMP(text, query)` menghasilkan tabel yang berisi indeks awal dan indeks akhir setiap ada bagian text yang sama dengan pattern. Terdapat juga *function* `def listToString(s)` yang menghasilkan hasil parameter list menjadi sebuah string.

```
#string matching dengan algoritma KMP
def algoritma_KMP(text, query):


'''=========================================='''
    def table_KMP_query(query):
        #panjang A == panjang query
        A = [0 for _ in range(0, len(query))]
        # inisialisasi
        A[0] = 0
        j = 0
        i = 1
        #pengisian tabel A
        while i < len(query):
            if query[i] == query[j]:
                A[i] = j + 1
                j += 1
                i += 1

            elif j > 0:
                j = A[j - 1]

            else:
                A[i] = 0
                i += 1

        return A

'''=========================================='''

    tabel_hasil = []
    table_kMP = table_KMP_query(query)
    i = 0  # indeks text
    j = 0  # indeks query

    #string  matching
    while i < len(text):

       if query[j] == text[i]:
           #ujung dari query, dimana text = query
           if j == len(query) - 1:
              # (indeks mulai, indeks selesai)
              ketemu = (i + 1 - len(query), i + 1)
              tabel_hasil.append(ketemu)

              #mencari hasil lain
                i += 1
                j = 0
                continue
            i += 1
            j += 1

        #mengembalikan nilai j=0
        elif j > 0:
            j = table_kMP[j - 1]

        #mencari indeks selanjutnya pada text yang
        sama dengan indeks awal query
        else:
            i += 1

    return tabel_hasil
```

```
#convert list to string
def listToString(s):

    strr = ""
    for i in s:
        strr += i

    return strr

#DRIVER

#database mini (contoh aja) berisi kata-kata toxic
database = ["anjing","asu","babi","tai","tolol"]

#input query
query = input("query  :")

#convert str to list
output = list(query)

#hide kata-kata toxic
for i in range (len(database)):
    temp=[]
    if(algoritma_KMP(query,database[i])!=[]):
       temp = ((algoritma_KMP(query,database[i])))
       for i in range (len(temp)):
            #batas bawah
            awal = temp[i][0]

            #batas atas
            akhir = temp[i][1]

            for i in range (awal,akhir):
                output[i] = "*"

#print output
outputt = listToString(output)
print("output :",outputt)
```

Pada contoh pengujian implementasi diatas, terdapat driver sederhana yang menguji fungsi pengecekan pattern pada text di atas. Diberikan sebuah database mini, yaitu sebuah array of *character*__sederhana sebagai contoh database yang menampung kata- kata yang perlu di *hide.*

Pada implementasinya, query yang di input oleh pengguna *game online* sebagai pesan di dalam g*ame* tersebut diproses terlebih dahulu sebelum tercetak ke layar. inputan menjadi text dalam fungsi algoritma KMP, dan akan dicek setiap elemen database apakah ada yang sama, jika ada maka akan di *hide* dengan *character* '*".

Berikut hasil pengujian implementasi di atas:

```
query  :asu kau tololllll!!!!!
output : *** kau *****lll!!!!!
```

```
query  :taitaitaiksdnlk???,.taitai7t237!#$
output : *********ksdnlk???,.******7t237!#$
```

```
query  :dasar babiiiii.....!!!!!!!!!!
output : dasar ****iiii.....!!!!!!!!!!
```

```
query  :bocah gak usah main anjing, beban
output : bocah gak usah main ******, beban
```

## IV. KESIMPULAN

Aplikasi algoritma pencocokan string memiliki banyak sekali manfaatnya. Salah satunya dalam *game online.* Untuk melakukan *hide* terhadap kata-kata *toxic* di dalam *game online* memerlukan algoritma *string matching* yang dapat mendeteksi kata tersebut *toxic* atau tidak. Diantara algoritma Brute Force, Knuth-Morris-Pratt, dan Boyer-Moore, algoritma Boyer-Moore menjadi yang paling mangkus/efisien dalam melakukan pengecekan dalam *string matching*.

## V. PENUTUP

Penulis bersyukur kepada Tuhan YME yang telah memberikan kesehatan dan keselamatan sehingga penulis mampu menyelesaikan makalah ini dengan tepat waktu. Penulis juga ingin berterima kasih kepada seluruh dosen pengampu mata kuliah Strategi Algoritma yang telah memberikan banyak ilmu kepada penulis, khusunya Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen mata kuliah IF2211 Strategi Algoritma K02 yang telah memberikan materi untuk penulisan makalah ini. Penulis juga menyampaikan terima kasih kepada teman-teman penulis dan juga semua orang yang telah membantu dalam penyelesaian makalah ini.

## REFERENSI

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

[2]https://id.wikipedia.org/wiki/Algoritme_pencarian_string#:~:text=3%20Lihat%20pula-,Algoritme%20brute%20force%20dalam%20pencarian%20string,pembanding%20dan%20studi%2Dstudi%20lainnya.

[3]https://translate.google.com/translate?u=https://en.wikipedia.org/wiki/Knuth%25E2%2580%2593Morris%25E2%2580%2593Pratt_algorithm&hl=id&sl=en&tl=id&client=srp&prev=search

[4] https://id.wikipedia.org/wiki/Algoritme_Boyer-Moore

## 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, 11 Mei 2021

Roy H Simbolon

13519068