Analisis Forensik Sidik Jari pada Investigasi Kejahatan Menggunakan Algoritma Pencocokan String

Sekar Larasati Muslimah (13517114)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13517114@std.stei.itb.ac.id

***Abstract*—Identifikasi identitas korban dan pelaku tindak kriminal dilakukan dengan menggunakan analisis *digital fingerprint*. Sidik jari setiap orang berbeda satu sama lain. Hal ini membantu proses penyelidikan tindak kriminal untuk mengungkap data diri korban dan pelaku. Proses penyelidikan identitas dapat dilakukan dengan mencocokan *fingerprint* yang ditemukan dengan database tersimpan. Salah satu cara yang dapat dilakukan untuk membantu pencocokan data *fingerprint* adalah dengan menerapkan algoritma pencocokan string. Makalah ini akan membahas penerapan algoritma pencocokan string pada proses analisis forensik berbasis sidik jari guna mengungkapkan identitas korban dan pelaku kejahatan.**

Keywords—Pencocokan String; ; Sidik jari; Digital fingerprint

I. PENDAHULUAN

Untuk mengungkapkan motif dan menentukan tindak lanjut atas suatu tindak kejahatan pihak kepolisian melakukan penyelidikan. Proses penyelidikan meliputi pengumpulan barang bukti, olah tempat kejadian perkara dan analisis forensik kepada korban-pelaku.

Identitas dan rekam jejak pelaku serta korban sangat berguna untuk mengungkapkan suatu tindak kejahatan. Salah satu cara untuk mengungkap identitas pelaku dan korban adalah melalui analisis forensik. Analisis forensik dilakukan dengan mengidentifikasi sidik jari yang ditinggalkan pelaku di tempat kejadian perkara dan sidik jari korban (korban yang dimaksud adalah korban meninggal tanpa identitas) jika masih memungkinkan untuk mengambil sampel sidik jari korban.

Petugas penyelidikan akan mengambil sampel sidik jari. Sampel sidik jari tersebut akan diubah menjadi suatu formula melalui pengolahan citra, thresholding, dan binerisasi gambar. Formula yang dihasilkan adalah dalam bentuk pola biner.

Setelah mendapatkan pola biner, dilakukan pencocokan pola biner sidik jari dengan database tersimpan. Dalam permasalahan pencocokan sidik jari ini, algoritma pencocokan string tepat untuk diimplementasikan.

Pencocokan pola biner sidik jari dapat dilakukan dengan mengimplementasikan algoritma pencocokan string khususnya algoritma *exact string matching*. Algoritma *exact string matching* tepat diterapkan karena pola sidik jari setiap orang berbeda satu sama lain. Pada makalah ini akan dibahas penggunaan algoritma pencocokan string khususnya algoritma exact string matching untuk mencocokkan pola biner sidik jari. *Algoritma yang dimaksud adalah algortima Knuth-Morris-Pratt, Boyer Moore, dan Regex (Regular Expression)*.

Gambar 1 : Forensic Analysis

https://www.cybrary.it/0p3n/ram-memory-forensic-analysis/

II. DASAR TEORI

A. *Pencocokan String*

Pencocokan string adalah pencarian string di dalam teks (string matching atau pattern matching). Persoalan pencarian string dirumuskan sebagai berikut.

Diketahui :

1. Teks, yaitu long string yang panjangnya n karakter.
2. Pattern, yaitu string yang panjangnya m karakter (m < n) yang akan dicari di dalam teks.

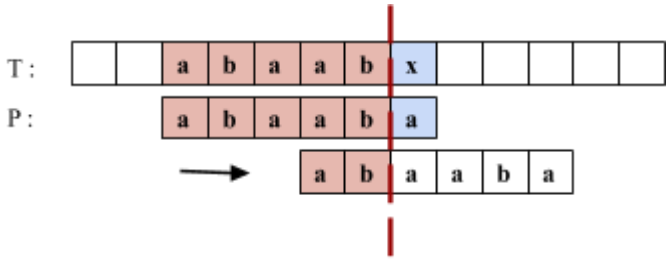
Pokok permasalahannya adalah menemukan lokasi pertama pattern bersesuaian dengan teks.

B. *Knuth-Morris-Pratt Algorithm*

Algoritma Knuth-Morris-Pratt (KMP) mencocokkan pattern dengan text dari kiri ke kanan layaknya algoritma brute force. Yang membedakan algoritma KMP dengan brute force adalah proses pergeseran pattern.

Jika terjadi ketidak-cocokan antara text T dan pattern P sehingga T[i] ≠ P[j], pattern akan digeser sedemikian rupa berdasarkan informasi proper prefix dari P[0..j-1] yang juga merupakan suffix dari P[1..j-1].

Mekanisme pergeseran pattern :



i = 8 ; j = 5

1. Menentukan *largest prefix* : "abaab" yang juga merupakan *suffix* "abaab"

   largest prefix - *suffix* = ab , panjang = 2.

2. Mengubah nilai j : j = 2 , nilai j baru menyatakan posisi pada *pattern* untuk memulai perbandingan lagi.
3. Menggeser *pattern* sejauh : panjang(abaab) - panjang(ab), sehingga *pattern* "abaaba" digeser sejauh 5 - 2 = 3 pergeseran.
4. Melakukan perbandingan karakter pada *pattern* dimulai dari *P[j]* dengan *T[i]*

Ide pemrosesan string pada algoritma KMP adalah melakukan *preprocess* pada *pattern* untuk menentukan setiap kemungkinan *largest prefix* - *suffix* dengan menggunakan fungsi pinggiran *KMP* (*KMP Border Function*) sebelum dilakukan pencocokan *string pattern* dengan *text*.

KMP Border Function :

Diketahui : *pattern P*

j = posisi ketidak-cocokan pada pattern

k = posisi tepat sebelum ketidak-cocokan (k = j-1)

Border function b(k) didefinisikan sebagai ukuran dari *largest prefix* dari *P[0..k]* yang juga merupakan *suffix* dari *P[1..k]*. Pada implementasi algoritma, *b()* direpresentasikan dengan *array* seperti pada tabel berikut.

Pattern P = "abaaba" dengan 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

Tinjau nilai b(k) untuk k = 4.

b(4) = 2 berarti panjang *largest presfix* dari *P[0..4]* yang juga merupakan *suffix* dari *P[1..4]* adalah 2 yaitu "ab".

Kompleksitas Waktu Algoritma *Knuth-Morris-Pratt* :

Border function : *O(m)* dan pencarian *string* : *O(n)*.

Kompleksitas waktu algoritma *KMP* adalah *O(M+n)*.

Keuntungan penggunaan algoritma *KMP* :

Algoritma tidak butuh penelusuran *backward* pada input *text T* sehingga baik untuk pemrosesan file yang sangat besar yang dibaca dari file eksternal atau melalui *network stream*.

Kekurangan penggunaan algoritma *KMP* :

KMP tidak bekerja dengan baik seiring meningkatnya nilai *alphabet*.

C. *Boyer Moore Algorithm*

Algoritma pencocokan *string Boyer Moore* didasarkan pada 2 teknik, yaitu :

1. Teknik *looking-glass*, yaitu menemukan *P* di dalam *T* dengan penelusuran backward di dalam *P* dimulai dari karakter terakhir *P*.
2. Teknik *character-jump,* yaitu ketika terjadi *mismatch* pada *T[i] == x* dan karakter *P[j]* tidak sama dengan *T[i],* terdapat 3 kemungkinan sebagai berikut.

   Kasus 1

   Jika *P* mengandung x, dimanapun, lakukan shift *P* ke kanan sehingga posisi x di *P* bersesuaian dengan *T[i]*.

   Kasus 2

Jika *P* mengandung x, dimanapun tetapi *shift right* ke *last occurance* tidak memungkinakan, maka *shift P* ke kanan 1 karakter ke *T[i+1]*.

Kasus 3

Jika kasus 1 dan 2 tidak dapat diaplikasikan, maka *shift P* sehingga *P[0]* bersesuaian dengan *T[i+1]*.

Ide pemrosesan *string* pada algoritma *Boyer Moore* adalah melakukan *preprocess* pada *pattern* untuk menentukan *last occurance* setiap karakter *pattern* di dalam lingkup *pattern* dengan menggunakan fungsi *Last Occurance* sebelum dilakukan pencocokan *string pattern* dengan *text*.

Last Occurance Function :

L(x) didefinisikan sebagai indeks i terbesar sehingga *P[i]* == x, atau menghasilkan -1 jika tidak ada x di dalam pattern *P*. Biasanya *L()* disimpan dalam sebuah *array.*

Tinjau pattern *P* = "abacab"

A = {a, b, c, d}

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

Kompleksitas waktu algoritma *Boyer Moore* untuk *worst case* adalah *O(nm+A)*. Tetapi, *Boyer Moore* melakukan pemrosesan dengan cepat ketika *alphabet* (A) besar dan lambat ketika *alphabet* kecil.

D. Regex Algorithm

Regex atau *regular expression* adalah "*special sequence of character*" yang membantu mencocokan / menemukan *string* lain atau kumpulan *string* dengan menggunakan *syntax* khusus dalam suatu *pattern*. *Regex* merupakan sebuah *pattern* (atau *filter*) yang merepresentasikan kumpulan *string* yang cocok terhadap *pattern* tersebut.

Regular Expression in Python :

Terdapat 14 *metaCharacter* yang digunakan dalam modul *re* (*regular expression*). Masing-masing *metaCharacter* memiliki fungsi yang berbeda sebagai berikut :

" \ " : digunakan untuk memberi makna khusus character yang mengikutinya.

" [] " : merepresentasikan sebuah kelas karakter.

" ^ " : mencocokkan bagian awal.

" $ " : mencocokan bagian akhir.

" . " : mencocokkan karakter apapun kecuali newline.

" ? " : memcocokkan *zero or one occurence.*

" | " : *OR*.

" * " : *any number of occurrences (including 0 occurrences)*.

" + " : *one or more occurrences.*

" {} " : menunjukkan jumlah kemunculan regular expression sebelumnya untuk mencocokkan.

" () " : melampirkan sekelompok regular expression.

Fungsi - fungsi dalam modul ***re*** :

1. compile()

   Regular expression dikompilasi menjadi objek *pattern* yang memiliki berbagai *method* operasi seperti pencocokan *pattern* atau substitusi *string*. Fungsi compile() membuat kelas karakter *regular expression*.

   Contoh penggunaan :

```
import re

# compile() creates regular expression character class [a-e],
# which is equivalent to [abcde].
# class [abcde] will match with string with 'a', 'b', 'c', 'd', 'e'.

p = re.compile('[a-e]')

# findall() searches for the Regular Expression and return a list upon finding

print(p.findall("Aye, said Mr. Gibenson Stark"))
```

2. split()

   Memisahkan *string* dengan kemunculan karakter atau pola. Setelah menemukan pola itu, karakter yang tersisa dari *string* dikembalikan sebagai bagian dari *list* yang dihasilkan.

   Syntax : `re.split(pattern, string, maxsplit=0, flags=0)`

3. sub()

   Syntax : `re.sub(pattern, repl, string, count=0, flags=0)`

   Kata 'sub' pada nama fungsi merepresentasikan SubString, *pattern regular expression* tertentu yang dicari di dalam *string* yang diberikan (parameter *string* pada fungsi). Setelah menemukan *substring*

pattern diganti dengan *repl*, mengecek *count*, dan mempertahankan jumlah kemunculannya.

4. subn()

   Syntax : `re.subn(pattern, repl, string, count=0, flags=0)`

   Mirip dengan fungsi sub(). Yang membedakan keduanya adalah penyajian output. Fungsi subn() mengembalikan *tuple* dengan perhitungan total *replacement* dan string baru.

5. escape()

   Syntax : `re.escape(string)`

   Mengembalikan string disertai semua *non-alphanumerics backslashed*.

E. *Identifikasi Sidik Jari*

Kebutuhan akan sistem identifikasi terus bertambah seiring berkembangnya teknologi. Objek yang dijadikan parameter identifikasi pun mulai bertambah. Beberapa objek yang dapat dijadikan parameter identifikasi adalah karu magnetik, tanda tangan, sidik jari, wajah, iris, suara, hingga DNA. Identifikasi dengan menggunakan sidik jari sebagai parameter dianggap paling praktis, akurat dan aman. Sidik jari adalah bagian tubuh manusia yang unik. Sidik jari memiliki beberapa keunggulan, yaitu :

1. *Parennial nature*, yaitu guratan sidik jari melekat pada manusia seumur hidup.
2. *Immutability*, yaitu sidik jari seseorang tidak akan pernah berubah semasa hidupnya.
3. *Individuality*, yaitu pola sidik jari unik untuk setiap manusia.

Karena tiga keunggulan tersebut sidik jari dijadikan parameter identifikasi.

Penggunaan sidik jari untuk mengungkap pelaku tindak kejahatan pertama kali diusulkan oleh Henry Fauld, seorang dokter asal Skotlandia.

Metode mengenali sidik jari yang umum digunakan pada proses identifikasi adalah minutiae. Minutiae merupakan pola unik dari garis *(ridge)* dan spasi *(valley)*. Secara umum garis dan spasi pada sidik jari menghasilkan beberapa pola unik yang dinamakan singularitas yang dibagi menjadi 3 jenis, yaitu *arch, loop,* dan *whorl*.

Gambar 3 : *Fingerprint Pattern Classes*

https://forensicmedindonesia.wordpress.com/2018/03/28/metode-identifikasi-sidik-jari/

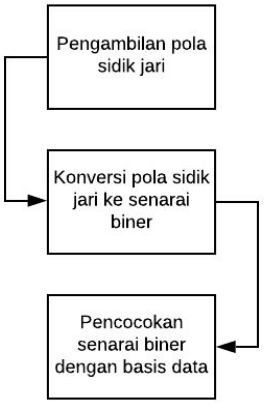
III. ANALISIS MASALAH DAN IMPLEMENTASI ALGORITMA

Proses identifikasi sidik jari pada proses analisis forensik diawali dengan pengumpulan sidik jari di tempat kejadian perkara. Pengambilan sidik jari korban dilakukan secara langsung pada jari korban dengan menggunakan tinta sidik jari dan kertas. Pengambilan sidik jari pelaku dilakukan dengan cara mengambil sidik jari yang tertinggal di tempat kejadian perkara, seperti di pintu, kaca, gelas, dll dengan menggunakan *powder* sehingga sidik jari nampak tetapi posisinya terbalik dari aslinya.

Proses selanjutnya adalah mengubah sidik jari ke dalam bentuk senarai biner melalui pemrosesan citra. Foto sidik jari akan diubah menjadi foto *grey scale* kemudian di konversi menjadi senarai biner melalui pemrosesan citra. Angka 1 menunjukkan bagian sidik jari yang menonjol dan angka 0 menunjukkan bagian sidik jari yang tidak menonjol (bukan garis sidik jari).

Senarai biner sidik jari akan dicocokkan dengan database tersimpan yang dimiliki oleh tim penyelidik perkara.

Skema pemrosesan sidik jari :



Gambar 2 : Skema proses analisis sidik jari

Pencocokan senarai biner sidik jari :

Setelah pola sidik jari dikonversi ke dalam bentuk senarai biner, dilakukan pencarian basis data yang bersesuaian dengan senarai biner sidik jari.

Dalam proses pencocokan dimungkinkan tidak ditemukan basis data sidik jari yang 100% bersesuaian dengan senarai biner sidik jari akibat *error* transformasi pola sidik jari menjadi pola biner.

1. Melakukan pengecekan kemiripan senarai biner sidik jari dengan semua basis data sidik jari. Saat melakukan pengecekan kemiripan dihitung persentase kemiripannya yaitu :

$$\%match = \frac{panjang\ pattern\ basis\ data\ tersimpan}{panjang\ pattern\ yang\ cocok} \times 100\%$$

2. Jika persentase kemiripan bernilai 100%, maka basis data yang berkaitkan akan langsung ditampilkan. Jika tidak 100%, akan ditampilkan semua basis data yang memiliki tingkat kemiripan 80%.

Implementasi algoritma pencocokan sidik jari :

```
def generateLPS(pattern, patSize, lps) :
    # panjang lps mula-mula 0
    lenLPS = 0
    # lps[0] akan selalu 0
    lps[0]
    i = 1
    while (i < patSize) :
        if (pattern[i] == pattern[lenLPS]) :
            lenLPS += 1
            lps[i] = lenLPS
            i += 1
        else :
            if (lenLPS != 0) :
                lenLPS = lps[lenLPS-1]
            else :
                lps[i] = 0
                i += 1
def KMP(pattern, text) :
    patSize = len(pattern)
    textSize = len(text)

    # inisialisasi array lps dengan ukuran yang sama dengan ukuran pattern
    # semua elemen array lps diinisialisasi dengan 0
    lps = [0]*patSize

    # membentuk array lps dari pattern
    generateLPS(pattern, patSize, lps)
    # print(lps)

    i = 0
    # hit : jumlah karakter terbanyak yang didapat dari pencocokan pattern dan teks
    hit =0
    # numMatch : jumlah pattern yang cocok dengan text diinisilisasi dengan 0
    numMatch = 0
    found = False
    # idx : indeks ditemukannya pattern pada text diinisialisasi dengan 0
    idx = 0
    while (i < textSize and not(found)) :
        if (pattern[numMatch] == text[i]) :
            i += 1
            numMatch += 1

        if (numMatch == patSize) :
            found = True
            idx = i-numMatch
            hit = numMatch
            numMatch = lps[numMatch-1]
        elif (i < textSize and pattern[numMatch] != text[i]) :
            if (numMatch != 0) :
                if(numMatch > hit):
                    hit = numMatch
                numMatch = lps[numMatch-1]
            else :
                i += 1
    return found,hit

def similarity(hit,length): #Mendapatkan %hit
    return hit/length*100

def loadJSON(filename):
    with open(filename) as data_file:
        data = json.load(data_file)
    return data

def search (faq, query) :
    for k, v in faq.items():
        found, hit = KMP(query, k)
        match = similarity(hit,15)
```

```
        if (match >= 80) :
            print(v)
            print(str(match)+"%")


faq = loadJSON('faq.json')
query = input(">> ")
search(faq, query)
```

Penerapan algoritma pencocokan sidik jari :

Diketahui basis data berisi :

Fingerprint	*Data*
010100110101000	DATA A
101000010001010	DATA B
111000101100010	DATA C
101110110010011	DATA D
011001010010101	DATA E
011011111001011	DATA F
110111000110000	DATA G
011110111111110	DATA H
011100111101100	DATA J
001110111011010	DATA I

Percobaan 1

Senarai sidik jari = 111000101100010

Analisis hasil :

```
sekar@sekar-Inspiron-14-3467:~/Desktop/Tubes-Stima-ChatBot$ python3 fi
ngerprint.py
>> 111000101100010
DATA C
100.0%
```

Senarai sidik jari bersesuaian dengan basis data fingerprint untuk data DATA C dengan kemiripan 100 %

Percobaan 2

Senarai sidik jari : 000011101100000

```
sekar@sekar-Inspiron-14-3467:~/Desktop/Tubes-Stima-ChatBot$ python3 fi
ngerprint.py
>> 000011101100000
sekar@sekar-Inspiron-14-3467:~/Desktop/Tubes-Stima-ChatBot$
```

Analisis hasil : tidak ditemukan fingerprint yang bersesuaian dengan senarai sidik jari sehingga program tidak menampilkan apapun.

Percobaan 3 :

Senarai sidik jari = 011110111111100

```
sekar@sekar-Inspiron-14-3467:~/Desktop/Tubes-Stima-ChatBot$ python3 fi
ngerprint.py
>> 011110111111100
DATA H
86.66666666666667%
```

Analisis hasil : tidak ditemukan fingerprint yang 100% cocok, tetapi ditemukan fingerprint dengan tingkat kecocokan 86.66666667% yang bersesuaian dengan data DATA H.

IV. KESIMPULAN

Berdasarkan hasil uji coba dari implementasi algoritma dan analisis dapat disimpulkan bahwa :

1. Algoritma pencocokan string dapat digunakan untuk melakukan pencocokan sidik jari yang telah dikonversi menjadi senarai biner dengan basis data sidik jari.
2. Algoritma pencocokan string dapat menampilkan data yang sesuai dengan senarai sidik jari yang memiliki tingkat kemiripin lebih dari 80%.

Masih diperlukan pengembangan program yang penulis buat untuk melakukan pencocokan senarai sidik jari. Program yang dibuat masih sangat sederhana dengan jumlah basis data yang masih sedikit.

REFERENCES

[1] https://www.omicsonline.org/proceedings/forensic-analysis-of-digital-fingerprint-based-biometric-data-20381.html diakses pada 25 April 2019 pukul 14.00

[2] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algortima. Program Studi Teknik Informatika ITB. 2018.

[3] Kurnia Rizqiani , M. Ramdhani , Achmad Rizal. Perancangan Perangkat Lunak Penghitung Rumus Sidik Jari Tipe Loop. Program Studi Teknik Informatika Institut Teknologi Telkom. 2009.

[4] https://forensicmedindonesia.wordpress.com/2018/03/28/metode-identifikasi-sidik-jari/ diakses pada 26 April 2019 pukul 09.06.

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, 26 April 2019

Sekar Larasati Muslimah - 13517114