Pattern Matching Implementation With Divide and Conquer Approach on Fingerprint Analysis in Detecting Fingerprint Pattern

Brian Albar Hadian – 13523048
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: author@gmail.com , author@std.stei.itb.ac.id

***Abstract*—Fingerprint pattern matching has already been used in many places. Authenticity caught using pattern matching usually used for significant and sometimes confidential purposes. Using string pattern matching, fingerprint pattern matching can be used to detect the authenticity between two fingerprints. This process can be streamlined by implementing divide and conquer algorithm. Analysis result will return the percentage of similarity between images. This implementation is hoped to reduce overhead of similar process during fingerprint pattern matching in demanding environment.**

Keywords—Pattern, KMP, Fingerprint, DnC, Matching

I. Introduction

Pencocokan sidik jari sudah banyak digunakan di banyak bidang kehidupan. Pencocokan sidik jari biasanya digunakan untuk berbagai kebutuhan. Pencocokan sidik jari cocok digunakan untuk kepentingan identifikasi dan verifikasi. Beberapa contoh diantaranya adalah sistem sekuritas, proses analisis forensik, pengamanan perbatasan negara, e-passports, dan sebagainya. Hal ini menjadi penting karena sifat dari sidik jari itu sendiri yang unik sehingga diperlukan algoritma yang tepat unutk menentukan kesamaan antar sidik jari.

Pencocokan sidik jari biasanya diterapkan dengan menggunakan algoritma yang didasarkan pada ilmu aljabar linear dan geometri. Penggunaan ini memanfaatkan analisis yang menggunakan sumber daya yang cukup banyak dan dapat mengakibatkan tingginya konsumsi daya apabila digunakan hingga tingkat tertentu. Dengan alasan itu, diperlukan adanya algoritma yang cukup mangkus dan dapat diandalkan untuk dijalankan pada lingkungan yang kritis.

Algoritm Knuth-Morris-Pratt merupakan algoritma pencocokan *string* yang digunakan untuk melakukan pencocokan secara *exact matching.* Algoritma ini biasanya digunakan dalam pencocokan string dalam pencarian kata kunci yang diperlukan. Beberapa contoh diantaranya adalah CV ATS, *search engine,* deteksi plagiarism, dan sebagainya. Algoritma ini dapat dimanfaatkan untuk deteksi kesamaan antara dua gambar dengan membandingkan piksel antara kedua gambar tersebut.

Algoritma Divide and Conquer merupakan algoritma penyelesaian yang memanfaatkan pendekatan pembagian masalah utama menjadi masalah yang lebih kecil dan melakukan proses penyelesaian yang sama. Algoritma ini biasanya digunakan dalam permasalahan yang dapat dibagi menjadi permasalahan yang lebih kecil dan memiliki proses penyelesaian yang sama untuk setiap subbagian. Beberapa contoh kasus diantaranya pengurutan (*sorting*), pencarian (*searching*), optimisasi, perkalian matriks, dan banyak lagi. Algoritma ini dapat dimanfaatkan untuk mempercepat proses perbandingan antara dua gambar dengan proses penyelesaian utama yaitu membandingkan nilai piksel antara kedua gambar.

II. Theory

A. Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) merupakan algoritma pencocokan *string* yang memanfaatkan sifat prefiks dan sufiks pada pola yang ingin diperoleh sedemikian sehingga dapat diperoleh untuk memperoleh jumlah total pergeseran minimum. Algoritma ini memiliki efisiensi terbaik ketika pola yang dicari memiliki kesamaan minimum dengan *string* yang dicari. Namun, algoritma ini memiliki efisiensi terburuk ketika pola yang dicari memiliki kesamaan maksimum dengan *string* yang dicari.

Algoritma ini bekerja dengan cara menentukan terlebih dahulu prefiks terpanjang yang juga merupakan sufiks dari pola tersebut. Tahap proses algoritma KMP adalah sebagai berikut.

1. Tentukan terlebih dahulu tabel *Longest Proper Suffix* (LPS) yang memberikan posisi pergeseran terkecil untuk melakukan proses pencocokan *string* kembali.
2. Dilakukan pencocokan karakter demi karakter
3. Apabila karakter pada *string* ke-i sama dengan karakter pada pola ke-j, tambahkan i dan j sebesar 1 satuan dan lakukan kembali pencocokan *string.*
4. Apabila karakter pada *string* ke-i tidak sama dengan karakteri pada pola ke-j, rujuk tabel LPS untuk memperoleh posisi yang dapat dilakukan proses pencocokan selanjutnya dan ulang nilai j menjadi 0.

5. Apabila nilai j sudah sama dengan panjang dari pola, maka telah ditemukan pola pada *string* dan proses pencarian selesai dilakukan.

Dalam kasus pencocokan piksel, Algoritma ini memiliki kompleksitas waktu sebesar O(m + n) untuk kasus terbaik dan kasus terburuk yang terdiri atas kompleksitas waktu pembentukan tabel LPS (O(m)) dan proses pencocokan piksel (O(n)) dengan m adalah panjang pola dan n adalah panjang baris gambar yang diperiksa.

B. Divide and Conquer (DnC)

Algoritma Divide and Conquer (DnC) merupakan algoritma yang digunakan untuk persoalan yang dapat dibagi menjadi persoalan yang lebih kecil dan setiap subbagian memiliki penyelesaian yang sama. Algoritma ini memiliki efisiensi terbaik ketika kompleksitas pembagian persoalan, kompleksitas penggabungan solusi, dan kompleksitas penyelesaian persoalan ketika persoalan sudah berukuran cukup kecil memiliki kompleksitas sekecil mungkin.

Bentuk dasar kompleksitas algoritma DnC adalah sebagai berikut.

$$T(n)\begin{cases} g(n) \\ T(n_1) + T(n_2) + T(n_3) + \cdots + f(n) \end{cases}$$

Tahap penggunaan algoritma DnC adalah sebagai berikut.

1. Tentukan terlebih dahulu proses penyelesaian untuk persoalan tersebut.
2. Bagi setiap persoalan menjadi persoalan dengan ukuran yang lebih kecil hingga mencapai suatu batasan tertentu.
3. Selesaikan setiap persoalan menggunakan proses penyelesaian yang telah ditentukan pada tahap (1)
4. Gabungkan solusi untuk setiap persoalan

Berdasarkan tahap penyelesaian persoalan dan bentuk dasar kompleksitas algoritma DnC, algoritma ini memiliki performa terburuk ketika setiap tahap membutuhkan proses yang memiliki kompleksitas eksponensial sehingga menghasilkan total kompleksitas sebesar $O(n^n)$.

Dalam kasus pencocokan *string* dalam gambar, algoritma DnC dapat memiliki total kompleksitas waktu sebesar O(pl) dengan p merupakan panjang dimensi dari gambar dan l merupakan lebar dimensi dari gambar.

III. METHODOLOGY

Dalam metode yang akan digunakan, data yang digunakan sebagai data percobaan adalah data gambar sidik jari berwarna dengan format .tiff, memiliki dimensi beragam, dan memiliki latar belakang putih.

A. Method

Proses pencocokan sidik jari dilakukan dengan memanfaatkan algoritma Knuth-Morris-Pratt (KMP) dan Divide and Conquer (DnC). Proses ini memanfaatkan sifat KMP yang memiliki kasus terbaik ketika pola yang dicari memiliki kesamaan minimum dengan *string* yang sedang diproses. Proses ini kemudian dipercepat dengan menerapkan algoritma DnC untuk membatasi pemrosesan hingga tingkat tertentu.

Diberikan dua buah gambar sidik jari (*reference image* dan *current image*), diterapkan langkah metode sebagai berikut.

1. Dilakukan pemrosesan awal (*preprocessing*) terhadap kedua gambar yang terdiri atas penyamaan dimensi antara kedua gambar sedemikian sehingga diperoleh koordinat awal dan dimensi yang akan diproses lebih lanjut.
2. Dibentuk suatu variabel root dengan struktur data QuadTree yang memiliki informasi mengenai koordinat absis awal dan akhir, koordinat ordinat awal dan akhir, dan empat struktur data QuadTree yang merepresentasikan empat bagian yang akan dibagi dari gambar tersebut, yakni top_left, top_right, bottom_left, dan bottom_right.
3. Pada awal proses, dilakukan perhitungan konversi *grayscale* untuk setiap piksel berdasarkan koordinat awal pada variabel root berdasarkan formula dibawah

   $$Grayscale = 0.299 * R + 0.587 * G + 0.114 * B$$

   dengan R adalah nilai piksel merah, G adalah nilai piksel hijau, dan B adalah nilai piksel biru pada piksel yang sedang diuji. Nilai ini kemudian diproses lebih lanjut sehingga diperoleh nilai deviasi *d* sebagai metode deteksi kesalahan.
4. Apabila nilai deviasi *d* dari *grayscale* memiliki nilai **diatas** batas nilai tertentu *g*, maka proses pembagian persoalan akan dilakukan dengan cara membagi dimensi root menjadi empat bagian utama, diantaranya,
   a. top_left yang mewakili bagian gambar atas kiri dengan dimensi tinggi sebesar tinggi variabel root dibagi dua;
   b. top_right yang mewakili bagian gambar atas kanan dengan dimensi tinggi sebesar tinggi variabel root dibagi dua;
   c. bottom_left yang mewakili bagian gambar bawah kiri dengan dimensi tinggi sebesar tinggi variabel root dibagi dua;
   d. bottom_right yang mewakili bagian gambar bawah kanan dengan dimensi tinggi sebesar tinggi variabel root dibagi dua;
5. Selanjutnya, untuk setiap subbagian yang diperoleh dari tahap (4), dilakukan pemrosesan seperti tahap (3) dan (4).
6. Apabila nilai deviasi *d* dari *grayscale* memiliki nilai **di bawah atau sama dengan** batas nilai tertentu *g*, maka proses pembagian persoalan akan dihentikan dan dilakukan proses pencocokan piksel untuk antara kedua gambar menggunakan algoritma KMP.
7. Untuk setiap kesamaan yang diperoleh, akan disimpan suatu total kesamaan yang merupakan jumlah kesamaan piksel yang ditemukan dan hasil akhir penjumlahan tersebut akan mengembalikan total kesamaan dibagi

dengan total piksel yang telah diuji kesamaannya. Nilai akhir ini selanjunya akan dirujuk sebagai *similarity ratio* (SR).

8. Nilai SR kemudian akan ditambahkan hingga proses berakhir dan dibagi dengan jumlah data yang diperoleh sehingga diperoleh rata-rata dari SR.
9. Nilai SR kemudian menjadi nilai yang akan merepresentasikan kesamaan yang diperoleh.

B. Implementation

```
function preprocessImage (current_image : RASTER, reference_image : RASTER) → QuadTree
KAMUS LOKAL
q : QuadTree
ALGORITMA
if current_image.height < reference_image.height then
    if current_image.width < reference_image.width then
        q.x_start ← 0
        q.x_end ← current_image.width
        q.y_start ← 0
        q.y_end← current_image.height
    else
        q.x_start ← 0
        q.x_end ← reference_image.width
        q.y_start ← 0
        q.y_end← current_image.height
else
    if current_image.width() < reference_image.width then
        q.x_start ← 0
        q.x_end ← current_image.width
        q.y_start ← 0
        q.y_end← reference_image.height
    else
        q.x_start ← 0
        q.x_end ← reference_image.width
        q.y_start ← 0
        q.y_end← reference_image.height
→ q
```

```
function checkBoundary (current_image : RASTER, root : QuadTree) → boolean
KAMUS LOKAL
    g : int = 245 { default value for greyscale boundary }
ALGORITMA
    mean ← 0
    i traversal root.y_start…root.y_end
        j traversal root.x_start…x_end
            mean ← mean + current_image.grayscale
    mean ← mean / ((root.y_end – root.y_start) * (root.x_end – root.x_start))
    deviation ← 0
    i traversal root.y_start…root.y_end
        j traversal root.x_start…x_end
            deviation ← deviation + absolute(current_image.grayscale – mean)
    deviation ← deviation / ((root.y_end – root.y_start) * (root.x_end – root.x_start))
    if (deviation > g) then
        → true
    → false
function determineSimilarity (current_image : RASTER, reference_image : RASTER, root : QuadTree) → float
KAMUS LOKAL
sr : float = 0
top_left : QuadTree
top_right : QuadTree
bottom_left : QuadTree
bottom_right : QuadTree
ALGORITMA
left_width_partition ← floor((root.x_end – root.x_start) / 2)
right_width_partition ← root.x_end – left_width_partition
left_height_partition ← floor((root.y_end – root.y_start) / 2)
right_height_partition ← root.y_end – left_height_partition
if checkBoundary(current_image, root) then
    root.top_left.x_start ← 0
    root.top_left,x_end ← left_width_partition
    root.top_left.y_start ← 0
```

```
    root.top_right.x_start ← right_width_partition
    root.top_right,x_end ← root.x_end
    root.top_right.y_start ← 0
    root.top_right.y_end ← right_height_partition
    root.bottom_right.x_start ← right_width_partition
    root.bottom_right,x_end ← root.x_end
    root.bottom_right.y_start ← right_heigth_partition
    root.bottom_right.y_end ← root.y_end
    root.bottom_left.x_start ← 0
    root.bottom_left,x_end ← left_width_partition
    root.bottom_left.y_start ← left_height_partition
    root.bottom_left.y_end ← root.y_end

    { recursive }
    sr ← sr + determineSimilarity (current_image,
reference_image, root.bottom_left)
    sr ← sr + determineSimilarity (current_image,
reference_image, root.bottom_right)
    sr ← sr + determineSimilarity (current_image,
reference_image, root.top_left)
    sr ← sr + determineSimilarity (current_image,
reference_image, root.top_right)
else
    { do pattern matching with KMP }
    → KMPMatching(current_image, reference_image,
root)

function KMPMatching (current_image : RASTER,
reference_image : RASTER, root : QuadTree) → int
KAMUS LOKAL
sameCount : int = 0
borderTable : map
getBorderTable(reference_image : RASTER, root :
QuadTree) → map
iterForPrefix, i, j, iterForSuffix
ALGORITMA
    borderTable ← getBorderTable(reference_image,
reference_tree);
    i traversal root.y_start..root.y_end
        iterForPrefix ← 0
        iterForSuffix ← 0
    while (iterForSuffix < root.x_end) do
        if (current_image.grayscale.suffix =
reference_image.grayscale.prefix) and (iterForPrefix =
(root.x_end - root.x_start) - 1)) then
            sameCount ← sameCount + 1;
            iterForPrefix ← 0;
            iterForSuffix ← iterForSuffix + 1;
        else if (current_image.grayscale.suffix =
reference_image.grayscale.prefix) {
            iterForPrefix ← iterForPrefix +  1;
            iterForSuffix ← iterForSuffix +  1;
        else if (iterForPrefix > 0) {
            iterForPrefix = borderTable[(i, iterForPrefix -
1)]
        else
            iterForSuffix ← iterForSuffix +  1
→ sameCount

procedure main (current_image : RASTER,
reference_image : RASTER) → float
KAMUS LOKAL
    totalSR : float = 0
ALGORITMA
    → determineSimilarity(current_image,
reference_image, preprocessImage(current_image,
reference_image))
```

IV. Result and Discussion

A. *Result*

1. Pencocokan sidik jari dengan data yang ada pada folder yang sama dengan nilai *g* sama dengan deviasi pada gambar referensi

   Gambar uji: test/DB2/102_2.tif

Hasil :

Average similarity : 0.0005716574906147081%

File with max similarity : test/DB2/105_1.tif with similarity 0.3295258955807878%

Time elapsed : 32234 ms

2. Pencocokan sidik jari dengan data yang ada pada folder yang sama dengan nilai *g* sama dengan ½ dari deviasi pada gambar referensi

Gambar uji: test/DB2/102_2.tif

Hasil :

Average similarity : 0.00013874901078604181%

File with max similarity : test/DB2/102_2.tif with similarity 0.23696682464454977%

Time elapsed : 29829 ms

3. Perbandingan pencocokan sidik jari antara penggunaan metode *brute force* dan metode awal dengan data yang ada pada folder yang sama dengan nilai *g* sama dengan dari deviasi pada gambar referensi

Gambar uji : test/DB2/102_4.tif

Hasil :

Average similarity : 0.3324966424448683%

File with max similarity : test/DB2/102_4.tif with similarity 100%

Time elapsed : 5598 ms

B. Discussion

Berdasarkan hasil pemrosesan pencocokan biasa, dapat dilihat bahwa hasil analisis kesamaan yang diperoleh dengan metode KMP dan DnC memiliki nilai yang cenderung kecil. Hal ini dapat disebabkan oleh berbagai faktor, diantaranya adalah faktor nilai batas deviasi *g* yang relatif besar, sifat data uji yang memiliki deviasi yang cukup kecil, dan fungsi KMP yang melakukan pergeseran piksel terlalu sering.

Faktor nilai batas deviasi *g* dapat memengaruhi hasil analisis karena faktor nilai ini menentukan apakah suatu struktur data QuadTree akan dibagi lebih lanjut menjadi QuadTree lain yang lebih kecil. Apabila nilai batas deviasi *g* memiliki nilai yang relatif besar, maka status QuadTree yang sedang diuji akan semakin cepat berhenti dan langsung melakukan perhitungan operasi KMP. Pada skala yang lebih besar, status QuadTree mungkin berhenti pada iterasi pertama sehingga melakukan analisis terhadap ukuran yang cenderung besar dan berakibat meningkatnya kompleksitas waktu. Dalam hal ini, nilai deviasi dapat ditentukan lebih lanjut yang disesuaikan dengan sifat dari data uji dan alat yang digunakan untuk mengambil data.

Selain itu, faktor data uji juga dapat memiliki pengaruh pada proses pencocokan nilai *grayscale* antara kedua gambar yang dibandingkan. Hal ini dapat dilihat dari data uji yang diberikan. Dengan dilakukan pemrosesan terlebih dahulu pada data uji, dapat ditentukan nilai deviasi pada gambar. Kemudian, setelah peninjauan ulang, kebanyakan dari data uji memiliki sifat yang cenderung memiliki deviasi yang cukup rendah. Hal ini dapat dibuktikan dengan banyaknya piksel yang cenderung hitam dibanding piksel yang berwarna putih sehingga memperkecil nilai deviasi. Berdasarkan pembahasan sebelumnya mengenai batas deviasi *g,* dapat dinyatakan bahwa analisis pencocokan sidik jari dapat memiliki nilai yang cukup besar apabila batas deviasi memiliki nilai yang cukup kecil atau disesuaikan dengan media dan sifat dari data uji itu sendiri.

Selanjutnya, sifat dari algoritma KMP itu sendiri dapat melakukan pergeseran saat dilakukan pencocokan sidik jari. Hal ini terjadi akibat sifat dari algoritma KMP yang akan melakukan pergeseran yang apabila terdapat nilai yang tidak sesuai. Selain itu, dengan nilai jangkauan dari skala *grayscale* yang cukup besar, hal ini memperkecil kemungkinan suatu piksel pada gambar referensi memiliki nilai yang sama dengan nilai piksel dengan gambar yang sedang diuji. Dengan meninjau bahwa gambar referensi merupakan gambar yang dikonversi menjadi gambar hitam putih, maka hal ini berakibat pada peningkatan kemungkinan ketidaksesuaian nilai antar piksel yang sedang diuji. Dengan pertimbangan tersebut, dapat dilihat bahwa penyebab utama untuk faktor ini adalah kecenderungan ketidaksesuaian untuk setiap piksel. Hal ini dapat diatasi dengan memberikan nilai batas kesalahan yang cukup sehingga dapat meningkatkan nilai similaritas dari algoritma.

Lalu, meninjau pada hasil kedua, dapat dilihat bahwa dengan memperkecil batas nilai deviasi *g*, tingkat similaritas dapat meningkat dan dapat diperoleh hasil yang lebih akurat dengan

sumber data. Hal ini sesuai dengan pembahasan sebelumnya bahwa nilai batas deviasi *g* memiliki pengaruh yang besar pada pencocokan sidik jari pada algoritma ini. Untuk pengembangan selanjutnya, diperlukan penelitan lebih lanjut untuk menentukan batas deviasi yang tepat untuk sumber data dengan kriteria yang diberikan.

Terakhir, berdasarkan hasil percobaan ketiga, dapat dilihat bahwa proses pencocokan sidik jari memiliki waktu yang relatif jauh lebih cepat dibandingkan algoritma yang diciptakan. Hal ini dapat diakibatkan oleh beberapa hal, diantaranya sifat gambar itu sendiri dan proses pencocokan

Pada akhirnya, berdasarkan kompleksitas waktu yang diberikan untuk setiap algoritma pada bab 2, dapat dilihat bahwa kompleksitas keseluruhan untuk proses pencocokan sidik jari yang dilakukan adalah sebagai berikut.

$$T(n)\begin{cases} O(m+n) \ , n=1 \\ 4T\left(\frac{n}{4}\right) + \ \ldots + n \end{cases}$$

Dengan menggunakan n = 4^k, diperoleh

$$T(n) = 4^k \, T(1) + \sum_{0}^{k} 4^k$$

$$T(n) = 4^k \, O(m+n) + \sum_{0}^{k} 4^k$$

$$T(n) = 4^k \, O(m+n) + \frac{4^{k+1} - 1}{3}$$

Lalu kembalikan n = 4^k, diperoleh

$$T(n) = n \, O(m+n) + \frac{n-1}{3}$$

Maka, dapat diperoleh bahwa kompleksitas *big-O* dari algoritma ini adalah

$$O(n) = n^2$$

VIDEO LINK AT YOUTUBE

https://youtu.be/TvF3W1wqQzk

APPENDIX

BrianHadianSTEI23/IdentiFi: A mini-project tackling the need of fingerprint identification based on known data for identifying purpose implemented using Pattern Matching and Divide and Conquer Algorithm. Made with love, by Brian A. Hadian (13523048).

ACKNOWLEDGMENT

Dengan ini, saya menyatakan terimakasih pada Dr. Nur Ulfa Maulidevi, S.T., M.Sc., selaku dosen pengampu Mata Kuliah IF2211 Strategi Algoritma, karena telah membimbing Saya hingga titik ini.

REFERENCES

[1] Munir, Umer & Javed, Muhammad. (2005). Fingerprint Matching using Ridge Patterns. Atherosclerosis. 116 - 120. 10.1109/ICICT.2005.1598565.

[2] Srinivas, I. V., Samnani, M., & Shaikh, M. S. (2018). *Study of string matching algorithm*. IOSR Journal of Computer Engineering (IOSR-JCE), 4th Somaiya International Conference on Technology and Information Management (SICTIM'18), 32–35. K J Somaiya Institute of Management Studies and Research (SIMSR). https://www.iosrjournals.org/iosr-jce/papers/Conf.17025-2017/Volume-1/7.%2032-35.pdf

[3] Munir, R. (2025). *IF2211 Strategi Algoritma – Semester II Tahun 2024/2025*. Departemen Informatika, STEI-ITB. Retrieved June 24, 2025, from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/stima24-25.htm

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, 24 Juni 2025

Brian Albar Hadian
13523048