# Aplikasi Program Dinamis dalam Menentukan Kemiripan Pola Tubuh Zebra dengan Algoritma Wagner-Fischer

William Fu - 13518055
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jalan Ganesha 10 Bandung*
*13518055@std.stei.itb.ac.id*

***Abstrak*—Setiap makhluk hidup, meskipun berasal dari spesies atau induk yang sama, pasti memiliki keunikannya masing-masing. Secara biologis, keunikan ini dapat dilihat dari sekuens rantai DNA setiap makhluk yang berbeda. Akan tetapi, perbedaan ini juga dapat dilihat dari karakteristik lainnya. Seperti sidik jari pada manusia, pola tubuh hitam-putih pada tubuh zebra juga bersifat unik – tidak ada satu pun zebra yang memiliki pola tubuh yang sama. Oleh karena itu, identifikasi pola dapat dilakukan dengan memanfaatkan algoritma program dinamis. Dalam makalah ini, penulis memanfaatkan konsep *Levenshtein distance* atau *edit distance* dalam menguji kemiripannya.**

***Kata kunci—algoritma, program dinamis, Levenshtein distance, edit distance, zebra***

## I. Pendahuluan

Ilmu komputasi memiliki banyak penerapan dalam berbagai bidang kehidupan manusia. Seiring perkembangan zaman, berbagai metode serta algoritma pemecahan masalah baru ditemukan dan dikembangkan. Keberadaan metode ini menjadi dasar berbagai penemuan baru atau pengembangan dari teknologi yang sudah ada.

Dalam rumpun ilmu zoologi, identifikasi binatang individual merupakan hal yang penting dalam melakukan penelitian. Berbagai cara identifikasi makhluk hidup dikembangkan dan ditemukan untuk mempermudah proses identifikasi makhluk hidup individual. Adanya penemuan ini diharapkan dapat membuat proses identifikasi/ penentuan kemiripan semakin efektif dan efisien. Tentunya penemuan ini diharapkan tidak mengurangi akurasi atau ketepatan dari identifikasi tersebut.

Salah satu pola identifikasi yang unik untuk spesies manusia (*Homo sapiens*) adalah keberadaan sidik jari yang berbeda-beda untuk setiap individu manusia. Akan tetapi, banyak spesies hewan lainnya pula yang memiliki karakteristik serupa. Koala (*Phascolarctos cinereus*), binatang marsupial endemik Australia, juga memiliki sidik jari unik sebagai cara identifikasi.

Selain itu, pada beberapa spesies, pola yang terdapat pada tubuh (*torso*) hewan juga dapat digunakan untuk kepentingan identifikasi. Pada harimau (*Panthera tigris*), contohnya, pola loreng jingga-hitam-putih yang ada pada tubuhnya ternyata bersifat unik. Selain itu, pola tubuh hitam putih pada tubuh zebra (*Equus sp.*) juga berbeda-beda untuk tiap individunya. Oleh karena itu, mulai dikembangkan aplikasi untuk memudahkan ilmuwan zoologi, penjaga cagar alam, dan pekerjaan terkait lainnya dalam melakukan identifikasi suatu hewan individual dalam kelompok.

Dalam makalah ini, metode program dinamis akan dimanfaatkan dalam menentukan kemiripan pola tubuh zebra. Aplikasi program dinamis ini akan didasarkan pada konsep *edit distance* (biasa disebut *Levenshtein Distance*) yang diperkenalkan oleh Vladimir Levenshtein pada tahun 1966.

## II. Dasar Teori

### A. Program Dinamis

Program dinamis (*dynamic programming*) merupakan algoritma pemecahan masalah dengan ciri khas pembagian suatu masalah menjadi sejumlah upamasalah. Metode ini mulanya ditemukan oleh matematikawan Richard Bellman sebagai cara untuk mengoptimasi proses pengambilan keputusan yang terdiri atas berbagai tahap (*multistage*). Jenis permasalahan yang diselesaikan dengan metode program dinamis umumnya memiliki upamasalah yang bisa saling tumpang tindih (*overlapping*). Sehingga, setiap upamasalah sebaiknya diselesaikan hanya sekali saja lalu direkam (biasa ke dalam suatu tabel). Melalui hasil yang direkam inilah, solusi global dapat diputuskan.

Algoritma ini ditandai dengan adanya tahap-tahap dalam pemecahan masalah dan digunakannya prinsip optimalitas. Dalam memecahkan masalah optimasi (*optimization problem*), akan dicari solusi terbaik untuk setiap upamasalah yang dihasilkan. Prinsip optimalitas dalam program dinamis menyebutkan bahwa suatu solusi yang paling optimal, akan menghasilkan solusi yang optimal pula untuk setiap upamasalah setiap tahapnya.

Dalam pemecahan masalah optimasi, letak perbedaaan metode program dinamis dengan algoritma *greedy* terletak pada upamasalah yang menjadi perhatian. Pada algoritma *greedy*, hanya satu upamasalah saja yang diperhatikan untuk sebagai solusi yang optimal, sedangkan program dinamis

memerhatikan seluruh kemungkinan upamasalah yang terjadi dan mengkomputasi solusinya secara rekursif.

Walaupun sama-sama memecah suatu masalah menjadi upamasalah secara rekursif, program dinamis memiliki perbedaan dengan algoritma *divide and conquer*. Letak perbedaannya terdapat pada jenis masalah yang ditangani program dinamis. Setiap upamasalah yang ditangani oleh program dinamis dapat tumpang tindih (*overlap*) satu dengan yang lainnya.

Terdapat dua jenis pendekatan dalam menyelesaikan masalah dynamic programming yaitu:

1. Program dinamis maju (*top down/ memoization*)
2. Program dinamis mundur (*bottom up/ tabulation*)

Perbedaan kedua pendekatan ini terletak pada tahap diinisiasinya algoritma, dengan pendekatan *top down* memulai komputasi pada subproblem yang paling kecil (tahap ke-1) hingga problem globalnya (tahap ke-*n*) sementara pendekatan *bottom up* adalah sebaliknya.

Seiring perkembangannya, program dinamis kini tak hanya menjadi metode penyelesaian masalah optimasi saja, melainkan juga permasalahan lainnya dalam bidang komputasi. Salah satu manfaatnya adalah dalam algoritma CYK (Cocke-Younger-Kasami) untuk menentukan apakah suatu *string* dapat diterima suatu tata bahasa bebas konteks (CFG/*Context Free Grammar*). Penerapan program dinamis lainnya akan dimanfaatkan dalam makalah ini.

## B. *Levenshtein Distance*

Pada tahun 1966, Vladimir Levenshtein, seorang ilmuwan Rusia, memperkenalkan konsep *edit distance*, yaitu banyaknya operasi *edit* minimum yang dibutuhkan untuk mengubah suatu *string* menjadi *string* lainnya. Terdapat tiga jenis edit yang dapat dilakukan, yaitu penyisipan (*insertion*) suatu karakter, penghapusan (*deletion*) suatu karakter, atau subtitusi suatu karakter (*subtitution*). Ketiga operasi edit tersebut dapat diilustrasikan sebagai berikut.

1. *Substitution*

| R | A | T | U |
|---|---|---|---|
| R | O | T | I |

**Gambar 1**. Perbandingan dua string. (*Sumber: milik penulis*)

Levensthein distance dari string “RATU” dan “ROTI” adalah dua. Pada ilustrasi diatas, karakter ‘A’ dan ‘U’ dapat disubtitusi dengan ‘O’ dan ’I’ agar kedua *string* sama.

2. *Deletion*

| H | U | T | A | N |
|---|---|---|---|---|
|  | U | T | A | N |

**Gambar 2**. Perbandingan dua string. (*Sumber: milik penulis*)

Levensthein distance dari string “HUTAN” dan “UTAN” adalah satu ditunjukkan oleh sel berwana *pink*. Pada ilustrasi di atas, karakter ‘H’ pada *string* “HUTAN” dapat dihapus agar kedua string sama.

3. *Insertion*

Melalui ilustrasi di atas pula, penambahan karakter ‘H’ pada string “UTAN” dapat dilakukan untuk membuat kedua *string* sama.

Misalkan *a* dan *b* merupakan string dengan panjang *m* dan *n* berurutan, maka *Levenshtein distance/ edit distance* antara kedua dinotasikan sebagai fungsi $\text{lev}_{a,b}(i,j)$, dengan *i* dan *j* merupakan panjang substring *a* dan *b* dari indeks nol. Fungsi ini akan dimanfaatkan dalam implementasi algoritma Wagner-Fischer.

$$\text{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \text{lev}_{a,b}(i-1,j) + 1 \\ \text{lev}_{a,b}(i,j-1) + 1 \\ \text{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

**Gambar 3**. Fungsi *Levenshtein Distance* (Sumber: https://www.cuelogic.com/blog/the-levenshtein-algorithm)

## C. *Algoritma Wagner-Fischer*

Ditemukan oleh Wagner dan Fischer pada tahun 1974, algoritma Wagner-Fischer merupakan salah satu metode untuk mencari *edit distance* dari dua buah string yang ada. Algoritma Wagner-Fischer memanfaatkan metode program dinamis dengan pendekatan *top down*. Untuk merekam setiap solusi *subproblem* yang ada, algoritma Wagner-Fischer membentuk sebuah matriks berukuran $m \times n$, dengan *m* dan *n* merupakan panjang kedua string yang ingin dibandingkan.

Dengan memanfaatkan fungsi yang terdapat pada subbab B, algoritma untuk menentukan *edit distance* dapat memanfaatkan metode program dinamis dengan *pseudocode* yang ada. Dalam hal ini, solusi dari permasalahan Levenshtein Distance dapat ditentukan pada sel matriks ke m, n. Komputasi untuk levenshtein distance dilakukan mulai dari sel 1,1.

**Levenshtein Distance Algorithm Pseudo-Code**

```
LevenshteinDistanceAlgo (Source, Target)
Int Distance = new Int [Source.Length + 1, Target.Length + 1)
If (Source.Lenght = = 0)
|       return Target.Length
End If
If (Target.Lenght = = 0)
|       return Source.Length
End If
For Int I=0 to Source.Length
|       Distance [I, 0] = I
End For
For Int J=0 to Target.Length
|       Distance [0, J] = J
End For
For Int I=1 to Source.Length
|       For Int J=1 to Target.Lengh
|       |       Int Cost = (Target [J - 1] = = Source [I - 1] ? 0 : 1
|       |       Distance [I, J] = Min (Min (Distance [I - 1, J ] + 1, Distance [I, J - 1] + 1),
|       |       Distance [I - 1, J - 1] + cost)
|       End For
End For
return Distance [Source.Length, Destination.Length]
```

**Gambar 4**. Pseudocode Algoritma Wagner-Fischer. (*Sumber: http://csjournals.com/IJCSC/PDF7-1/45.%20Jai.pdf*)

Sebagai ilustrasi, akan diambil dua buah kata: “kasur” dan “kasar” sebagai *string a* dan *b* yang ingin dibandingkan. Maka akan dibentuk matriks berukuran 5 × 5 sebagai berikut.

| | | K | A | S | U | R |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| K | 1 | | | | | |
| A | 2 | | | | | |
| S | 3 | | | | | |
| A | 4 | | | | | |
| R | 5 | | | | | |

**Gambar 5**. Matriks 5 × 5 (*Sumber: milik penulis*)

Pertama, akan diisi sel matriks pada (1,1), karena min(1,1) tidak sama dengan 0, maka akan dicari nilai minimum berdasarkan fungsi *edit distance*. Dengan demikian didapatkan nilai berikut.

$$lev_{a,b}(i-1,j)+1 = lev_{a,b}(0,1)+1 = 1$$

$$lev_{a,b}(i,j-1)+1 = lev_{a,b}(1,0)+1 = 1$$

$$lev_{a,b}(i-1,\ j-1)+0 = lev_{a,b}(0,0)+0 = 0$$

Pada persamaan ketiga, lev(0,0) tidak ditambah dengan 1 sebab karakter a[i] sama dengan b[j]. Nilai minimum antara ketiga persamaan ini adalah 0, sehingga sel matriks(1,1) akan diisi dengan nilai 0. Demikian selanjutnya untuk sel matriks (1,2), (1,3), dan selanjutnya hingga matriks terisi sebagai berikut.

| | | K | A | S | U | R |
|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 |
| K | 1 | 0 | 1 | 2 | 3 | 4 |
| A | 2 | 1 | 0 | 1 | 2 | 3 |
| S | 3 | 2 | 1 | 0 | 1 | 2 |
| A | 4 | 3 | 2 | 1 | 1 | 2 |
| R | 5 | 4 | 3 | 2 | 2 | 1 |

**Gambar 6**. Matriks 5 × 5 terisi (*Sumber: milik penulis*)

Memperhatikan sel matriks posisi 5,5 (ditandai kuning) pada Gambar 4, maka dapat ditentukan bahwa jumlah operasi edit yang dibutuhkan untuk mengubah string a menjadi b adalah 1 operasi saja. Hal ini dapat dibuktikan dengan mensubtitusi karakter ke-4 pada string a dan b (mengubah ‘u’ menjadi ‘a’ atau sebaliknya).

Melalui matriks berikut, juga dapat dilihat bahwa nilai suatu sel matriks(i,j) ditentukan oleh nilai tiga sel matriks yang tepat berada di kiri, atas, dan diagonal kiri sel tersebut. Sel ini pula yang menentukan jenis operasi edit apakah yang perlu dilakukan.

| | **i-1** | **i** |
|---|---|---|
| **j-1** | sub | Ins |
| **j** | del | |

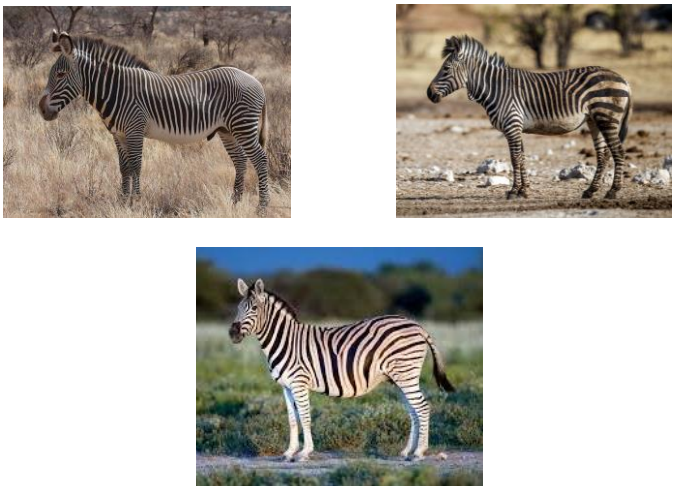
**Gambar 7**. Jenis operasi edit berdasarkan posisi realtif sel matriks (*Sumber: milik penulis*)

Melalui gambar di atas, jenis operasi yang sel pada posisi i, j-1 menunjukkan jumlah operasi *insertion* yang perlu dilakukan pada *string*, sedangkan sel pada posisi i-1, j-1 menunjukkan jumlah operasi *substitution* yang perlu dilakukan, selanjutnya sel pada i-1, j menunjukkan jumlah operasi *deletion* yang perlu dilakukan pada string.

### *D. Zebra*

Zebra merupakan binatang endemik Afrika yang berada di genus Equus. Spesies ini tersebar pada pegunungan serta dataran berumput di Afrika bagian timur dan selatan. Spesies zebra yang paling umum dijumpai adalah *Equus quagga*, sementara dua spesies zebra lainnya: *Equus zebra* dan *Equus grevyi* merupakan spesies zebra yang terancam punah (*endangered*).

Salah satu ciri khas yang dimiliki oleh zebra adalah pola tubuhnya yang berwarna hitam-putih. Pola tubuh ini berbeda untuk setiap spesies zebra yang berbeda pula. Selain itu, pola tubuh ini juga bersifat unik untuk setiap zebra individual, layaknya sidik jari (*fingerprint*) pada manusia. Sampai saat ini, masih belum ditemukan alasan dari karakteristik ini.



**Gambar 8**. Berbagai zebra dari spesies berbeda (dari kiri ke kanan) *Equus grevyi*, *Equus quagga*, *Equus zebra*. (*Sumber: https://upload.wikimedia.org/wikipedia/commons*)

## III. PEMBAHASAN

### *A. Ekstraksi Pattern dari Pola Tubuh Zebra*

Sebelum diolah dengan menggunakan algoritma Wagner-Fischer, pola tubuh Zebra pertama-tama akan diekstrasi dengan memanfaatkan modul Python Image Library (PIL) yang

disediakan oleh bahasa pemrograman Python. Melalui modul ini, dapat diketahui nilai RGB dari setiap pixel pada gambar. Nilai RGB ini kemudian dikonversikan menjadi tingkat *brightness*/kecerahan pixel. Hal ini dilakukan untuk mempermudah pembangkitan pattern nantinya, selain itu, pola tubuh zebra umumnya didominasi warna hitam, putih, atau abu-abu pada beberapa bagian kecil.

Sebagai contoh, penulis mengambil gambar pattern zebra sebagai berikut untuk diolah

**Gambar 9**. Pola Tubuh Zebra (*close-up*) (*Sumber:* [pixabay.com](pixabay.com))

Untuk setiap pixel pada gambar, tingkat kecerahannya akan direpresentasikan dengan sebuah karakter, dimulai dari karakter 'L' (ASCII value 76) hingga karakter 'Y' (ASCII value 89). Dengan demikian didapatkan string sebagai berikut

```
VUNRXSMWQRVOVRWQTWPVSRWQWPVQVSRR
VNOWQOWOTSQRUSPWPRUOWQVPURSUQURQ
XNNWQQWOVQPRWNSVNVQSRRRPUNTROVOO
WPNWSOWOWONWSMWRQUNVOTNUQNWPOWNN
VRMSTMTTTNRXPQWPSROVOSPUNOWNOVMN
SVNQVMQWQMTVNQVOVNTTQOURMSVMQUNN
OVRNVNRXONVVMSTQSNVQQQWNMVTLSTOO
NRWOSPPXPNUTMSUUMQXONVSMPWPMVSMT
NNUVOOOWRNUSNRWPNTTMQWPMTVMNWSMW
QMQWRMNWRNUSMSUMOVRMUWNOWTLRWQMW
UNNVVMOWQNVQMUSMSVNNWTNRXPMVWNMT
WPNSWNNWPNWPNWPMVTMRWPMVWLPXVMNR
WQMPWONWOOWNPWNNWQNUVNOWQMUWQMQN
WSMOWPNWNPVMRUMQWONWSMTVNOWVMMVM
```

**Gambar 10**. Pattern hasil ekstraksi pola tubuh zebra (*Sumber: milik penulis*)

Fungsi pengekstrakan pattern ini diimplementasikan oleh penulis dalam bahasa Python dengan fungsi `convert_image_to_string` sebagai berikut.

```python
from PIL import Image

def convert_image_to_string(imagename):
    '''
    Konversi image file dengan nama imagename
    ke string berdasarkan brightness
    '''

    # Menginisialisasi tabel mapping dari
    # tingkat kecerahan pixel ke sebuah karakter
    b_to_c = init_brightness_mapping()

    image = Image.open(imagename).convert("RGB")

    pixels = list(image.getdata())
    brightness = []
    for pixel in pixels:
        brightness.append(int(sum(pixel)/3))

    colorchar = []
    for bright in brightness:
        colorchar.append(b_to_c[bright])
    colorstring = ""

    return colorstring.join(colorchar)
```

**Gambar 11**. Implementasi ekstraksi pola tubuh zebra dalam bahasa Python (*Sumber: milik penulis*)

### *B. Implementasi Algoritma Wagner-Fischer*

Algoritma Wagner-Fischer diimplementasikan dalam bahasa Python sesuai dengan pseudocode (lihat Gambar 2). Dalam implementasi ini, library Numpy dari Python dimanfaatkan untuk mempermudah perhitungan dalam matrix yang telah disusun.

```python
import numpy as np

def lev(string_a, string_b):
    '''
    Implementasi algoritma Wagner-Fischer
    dalam mencari edit distance dua buah string
    '''
    # Menginisialisasi variabel
    size_x = len(string_a) + 1
    size_y = len(string_b) + 1
    matrix = np.zeros((size_x, size_y)) # matrix m x n

    # Mengimplementasikan basis
    # if min(i,j) = 0 --> matrix(i,j) = max(i,j)
    for x in range(size_x):
        matrix [x, 0] = x
    for y in range(size_y):
        matrix [0, y] = y

    # Memulai pengisian tabel
    # Pada tahap ini,
    for x in range(1, size_x):
        for y in range(1, size_y):

            offset = 1

            # Jika karakter sama, maka offset
            # untuk sel diagonal di set 0
            if string_a[x-1] == string_b[y-1]:
                offset = 0

            matrix [x,y] = min(
                matrix[x-1,y] + 1,
                matrix[x-1,y-1] + offset,
                matrix[x,y-1] + 1
            )

    # Edit distance berada pada sel
    # paling pojok atas
    return (matrix[size_x - 1, size_y - 1])
```

**Gambar 12**. Implementasi algoritma Wagner-Fischer dalam bahasa Python (*Sumber: milik penulis*)

Untuk mencoba fungsi yang dirumuskan di atas, penulis menggunakan kasus uji seperti ilustrasi pada bab II, dalam hal ini dua buah *string* yang akan dibandingkan adalah "KASUR" dan "KASAR". Hasil pengujian program dapat dilihat pada Gambar 9 di bawah ini.

```
PS C:\Users\ASUS\Desktop\makalah> python zebra.py
String pertama    : KASUR
String kedua      : KASAR
-- TAMPILAN MATRIX --
[[0. 1. 2. 3. 4. 5.]
 [1. 0. 1. 2. 3. 4.]
 [2. 1. 0. 1. 2. 3.]
 [3. 2. 1. 0. 1. 2.]
 [4. 3. 2. 1. 1. 2.]
 [5. 4. 3. 2. 2. 1.]]
Jumlah operasi edit yang dibutuhkan adalah : 1 buah
```

**Gambar 13**. Hasil perbandingan dua buah string dengan fungsi lev(string_a, string_b) (*Sumber: milik penulis*)

### *C. Menentukan Kemiripan Pola Tubuh Zebra*

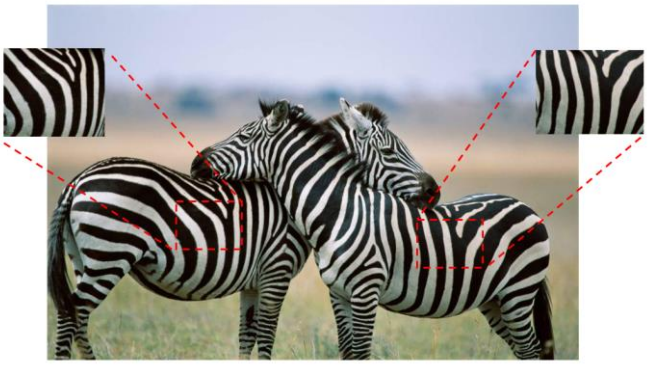
Dalam menentukan tingkat kemiripan pola tubuh zebra, penulis memanfaatkan *edit distance* yang diperlukan per pixel gambar yang digunakan sebagai rasio kemiripan antara dua buah pola yang ditemukan. Kemiripan dua buah gambar (dalam persen) dapat ditentukan dengan persamaan berikut.

$$ratio = \left(1 - \frac{lev(a,b)}{|a|}\right) \times 100\%$$

Dengan demikian, langkah pengujian untuk menguji pola tubuh zebra yang berbeda adalah sebagai berikut.

1. Melakukan ekstraksi *brightness* pada setiap pixel gambar, lalu memetakannya menjasi sebuah sekuens string.
2. Menjalankan algoritma Wagner-Fischer dengan parameternya adalah dua buah *string* hasil ekstraksi gambar.
3. Menghitung rasio kemiripan kedua buah gambar dengan sesuai dengan persamaan di atas.

### *D. Hasil Analisis*



**Gambar 14**. Dua ekor zebra (*Sumber: https://www.despines.com /2008/10/15/zebra-hug/?doing_wp_cron=1588328210. 8097529411315917968750)*

Untuk melakukan analisis kemiripan dua ekor zebra, penulis memilih gambar dua zebra yang identik berikut dan mengambil pola tubuh kedua zebra tersebut sebagai sampel. Hasil ekstraksi pattern kedua zebra tersebut disajikan pada Tabel 1 sebagai berikut.

**Tabel 1.** Hasil Ekstraksi Pola Tubuh Zebra

| Nama | Gambar | Ekstraksi |
|---|---|---|
| *zebra1* |  | NOOUWWPNNNQWXTNNNNQWUPMMMMMQTRONN NOVWROOOSUPOOOOOOPTXXWWTPOOOONNNN NONNUWWPNNNNPWXWONONOVWVPMMMMMQUTO NNNSWUOOOOOOOOOPRSVWWVSPONNNNNNNNN NONNOVWWONNNOVXWQNNNNSWWUNMMMMMRV UONNPVVQNOOOOOOQVWWWWUPNNNNNNNNNN NOSOOPWWWPN . . . |
| *zebra2* |  | QQQOMMMMMMMNQRQOMMMMMMNPQOMMMMMNP PNMMMMMMPOMMMMOOMMMMMMMMNMMMMMMN NQQQQOMMMMMMMNQRQPNMMMMMMOQQNMMMM MNPPOMMMMOQNMMMMPOMMMMMMMNNMMMMMM NOPQQRQOMMMMMMMNPRRQOMMMMMMPQPOM MMMMNPQNMMPQNMMMMNQPMMMMMMOOMMMMM MNOMOQRRQOM . . . |

Kemudian, kedua teks ini akan dibandingkan dengan algoritma Wagner-Fischer untuk dicari *edit distance*-nya. Untuk memudahkan komputasi, ukuran kedua gambar (*zebra1* dan *zebra2*) diseragamkan. Ukuran gambar yang dimanfaatkan adalah 67 × 52 pixel (*width* × *height*), sehingga panjang teks yang dihasilkan memiliki 3484 karakter.

Melalui fungsi Levenshtein yang telah diimplementasikan, didapatkan data sebagai berikut.

```
PS C:\Users\ASUS\Desktop\makalah> python zebra.py
Membandingkan zebra1.png dengan zebra2.png . . .
JUMLAH KARAKTER  : 3484 karakter
EDIT DISTANCE    : 1573
RASIO KEMIRIPAN  : 54.85 %
WAKTU            : 31.83 detik
```

**Gambar 15**. Hasil perbandingan kedua pola zebra (*Sumber: milik penulis)*

Edit distance yang dihasilkan dari perbandingan kedua buah gambar adalah 1573. Dengan demikian, rasio kemiripan kedua sampel pola zebra adalah 54.85 persen.

Pola pengujian ini dapat dimanfaatkan untuk mengukur kemiripan pola tubuh zebra dari suatu spesies (misal, *Equus zebra*) dengan spesies zebra lainnya (misal, *Equus grevyi*) untuk membantu menentukan relasi kedua spesies tersebut.

Selain itu, algoritma ini juga dapat dimanfaatkan dalam mengidentifikasi zebra pada suatu cagar alam. Seluruh pola tubuh zebra dapat direkam pada basis data dan dibandingkan dengan foto/ rekaman yang diambil. Apabila tingkat kemiripannya sesuai parameter (> 80% misalnya), maka dapat diketahui identitas zebra tersebut.

### *E. Analisa Kompleksitas*

Algoritma Wagner-Fischer memiliki kompleksitas waktu sebesar O(m×n) dengan m dan n merupakan panjang masing-masing string yang dibandingkan. Hal ini disebabkan nilai setiap harus ditentukan terlebih dahulu agar dapat menyelesaikan. Operasi penentuan nilai pada sel ini memiliki kompleksitas waktu konstan O(1).

Sedangkan, kompleksitas ruang (*space complexity*) algoritma juga sebesar O(m×n). Hal ini disebabkan dibutuhkannya matrix dengan sebesar m x n sesuai dengan panjang kedua string.

Berikut ini adalah tabel perbandingan waktu berjalannya algoritma Wagner-Fischer dengan ukuran gambar *zebra1* dan *zebra2* yang berbeda.

**Tabel 2.** Perbandingan *runtime* program untuk ukuran gambar berbeda

| Ukuran (w×h) | Jumlah pixel | Waktu (s) | Rasio (%) |
|---|---|---|---|
| 67 × 52 | 3484 | 31.83 | 54.85 |
| 100 × 78 | 7800 | 230.29 | 56.69 |
| 134 × 104 | 13936 | 516.86 | 57.56 |

## IV. Penutup

### A. Kesimpulan

Metode program dinamis memiliki banyak sekali aplikasi dalam berbagai disiplin ilmu. Dengan konsep edit distance, dapat ditentukan kemiripan antara dua buah pola. Konsep ini kemudian dapat dimanfaatkan untuk menentukan kemiripan pola dalam bentuk gambar pula. Hal ini dapat dilakukan dengan mengkonversikan setiap pixel pada gambar menjadi sebuah string yang sesuai berdasarkan parameter yang diinginkan (dalam hal ini tingkat kecerahan). Dengan demikian, rasio kemiripan dua buah gambar dapat ditentukan melalui perbandingan edit distance dengan panjang teks pola tubuh zebra secara keseluruhan.

### B. Saran

Penentuan kemiripan kedua pola tubuh zebra dapat dilakukan dengan pendekatan lainnya (selain program dinamis). Algoritma program dinamis yang ada juga dapat digabungkan dengan algoritma lainnya (misal *Divide and Conquer*) untuk mengurangi kompleksitas waktu yang ada. Selain itu, algoritma *pattern matching*, seperti algoritma Knuth-Morris-Pratt, atau algoritma Boyer-Moore dapat dimanfaatkan pula untuk menentukan kemiripan pola.

## Link video youtube

Penulis juga membuat video mengenai makalah ini yang dapat diakses pada tautan berikut https://youtu.be/WPUlA1M3MkE.


## Ucapan terima kasih

Pertama-tama, penulis ingin menghaturkan puji dan syukur kepada Tuhan Yang Maha Esa sebab atas rahmat dan berkat-Nya, makalah ini dapat diselesaikan dengan baik dan lengkap. Penulis pula ingin mengucapkan terima kasih kepada Ibu Dr. Masayu Leylia Khodra, ST., MT., Ibu Dr. Nur Ulfa Maulidevi, ST., M.Sc., Bapak Dr. Ir. Rinaldi, MT selaku dosen mata kuliah IF2211 Strategi Algoritma atas bimbingannya selama satu semester perkuliahan ini. Tak lupa juga penulis ingin mengucapkan terima kasih pula kepada seluruh pihak yang terlibat dalam penyelesaian makalah ini.


## Referensi


[1] Levitin, A. (2012). Introduction to The Design & Analysis of Algorithms. Boston: Pearson.

[2] Jones, N. C., Pevzner, P. A., & Pevzner, P. (2004). An Introduction To Bioinformatics Algorithms. MIT press.

[3] Gilleland, Michael. (2006), Levenshtein Distance, in Three Flavors. University of Pittsburgh (diakses melalui https://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Spring2006/assignments/editdistance/Levenshtein%20Distance.htm pada 1 Mei 2020 pukul 19:11 WIB)

[4] World Animal Foundation. Zebra Fact Sheet. (diakses melalui http://www.worldanimalfoundation.net/f/Zebra.pdf pada 1 Mei 2020 pukul 13:21 WIB)

[5] IF2211 Strategi Algoritma (2020). Program Dinamis (*Dynamic Programming*). KK Informatika STEI ITB. (diakses melalui https://drive.google.com/drive/folders/1Y-iWGq1ez1XxsAMCREOkKFvpy2roehWj pada 29 April 2020 pukul 11:22 WIB)

[6] Levensthein Distance (2019) (Diakses melalui https://devopedia.org/levenshtein-distance pada 1 Mei 2020 pukul 20:00 WIB)

[7] National Sun Yat-sen University. Wagner-Fischer Algorithm(Edit-Distance, 1974) (diakses melalui http://par.cse.nsysu.edu.tw/~lcs/Introduction.php pada 1 Mei 2020 pukul 19:32 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.

Tangerang Selatan, 2 Mei 2020

William Fu
13518055