Penerapan Algoritma Vektor pada *Finite Automata* untuk Perkiraan Kecocokan String DNA Aves

Yessica Nataliani

Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Jl. Diponegoro 52-60, Salatiga 50711, Indonesia
Email: yessica_24@yahoo.com

Abstract

Finite automata is a very useful device to calculate string. An approximate string matching problem is to find the location where a pattern is approximately matched with sub string taken from some texts with appointed maximum difference. We use bit vector algorithm to solve approximate string matching problem. Bit vector algorithm is used properly by using parallelism between the text and the pattern, because vector parallelism can take a short time at execution process by reducing iteration. Approximate string matching problem can be implemented in biology when we test the DNA matching, in example for Aves DNA.

Key Words: Approximate String Matching Problem, Bit Vector Algorithm, DNA Matching.

1. Pendahuluan

McCulloch dan Pitts [1] mengemukakan mesin abstrak sederhana yaitu *finite automata* untuk memodelkan *neuron nets*. *Finite automata* dapat dinyatakan sebagai model pengenal bahasa. Bahasa yang dikenal *finite automata* adalah bahasa sederhana. Ada dua macam *finite automata* yaitu *deterministic finite automata* dan *non deterministic finite automata*.

Finite automata merupakan alat abstrak yang berguna untuk mengkomputasi suatu string. Sebagai contoh, automata dapat dipakai untuk komputasi biologi yaitu mencocokkan string DNA. Pada artikel ini akan diangkat permasalahan untuk mencocokkan string DNA pada Aves.

Salah satu cara untuk mempercepat proses komputasi adalah dengan memperlakukan string sebagai vektor pada algoritma vektor yang dikerjakan oleh *finite automata*.

Hasil penelitian Bergeron dan Hamel menyatakan bahwa algoritma vektor dapat digunakan untuk komputasi suatu output vektor $r = r_1\ r_2\ \ldots\ r_m$ dari suatu input vektor $e = e_1\ e_2\ \ldots\ e_m$, dengan m adalah panjang vektor [2] [3].

Selain itu, hasil penelitian Huson (2004) menyatakan algoritma vektor bit dapat mempercepat proses komputasi untuk memperkirakan kecocokan

string [4].

Aplikasi *approximate string matching* salah satunya dalam bidang biologi, yaitu untuk permasalahan uji kecocokan DNA. Informasi tentang DNA diperoleh dari internet maupun buku-buku yang berhubungan dengan DNA, seperti buku karangan Pratiwi, dkk. [5] maupun Donald [6], sedangkan untuk database DNA yang digunakan untuk uji kecocokan DNA diambil dari DDBJ Database [7].

2. Kecocokan String

Permasalahan dasar dalam mencocokkan string (*string matching*) adalah menemukan pola teks tertentu yang diinginkan dari suatu teks. Beberapa variasi dalam permasalahan mencocokkan string adalah menghitung jumlah ketidakcocokan, menghitung jumlah penyisipan, menghitung jumlah penyisipan dan penghapusan, menghitung jumlah ketidakcocokan, penyisipan, penghapusan, dan mengukur ketergantungan bahasa (fonetik, morfim, dan lain-lain).

Kecocokan string dibagi menjadi dua bagian yaitu mencocokkan string dengan pasti (*exact string matching*) dan mencocokkan string dengan perkiraan (*approximate string matching*). Di dalam penulisan ini hanya akan dibahas mencocokkan string dengan perkiraan (*approximate string matching*). Perbedaan dasar antara keduanya adalah pada *exact string matching*, substring dari teks yang dicari harus betul-betul sama dengan pola yang diberikan, baik panjangnya maupun susunan karakternya. Sedangkan pada *approximate string matching*, substring dari teks yang dicari tidak perlu sama persis dengan pola yang diberikan, bisa panjangnya yang berbeda, bisa pula susunan karakternya yang berbeda. Algoritma yang sering digunakan untuk mencocokkan string dengan pasti adalah Algoritma Knuth-Morris-Pratt, Algoritma Boyer Moore, dan Algoritma Brute Force, sedangkan untuk mencocokkan string dengan perkiraan digunakan *edit distance* dan algoritma vektor yang akan dibahas pada penulisan ini.

3. Algoritma *Edit Distance* untuk Perkiraan Kecocokan String

Permasalahan memperkirakan kecocokan string (*approximate string matching*) adalah menemukan semua lokasi di mana suatu pola tertentu (dengan panjang *m*) diperkirakan cocok dengan substring dari suatu teks yang diketahui (dengan panjang *n*) dengan maksimal perbedaan tertentu.

Perhitungan untuk memperkirakan kecocokan string biasanya menggunakan *edit distance*, yang meliputi tiga operasi dasar yaitu penyisipan (*insertion*), penghapusan (*deletion*), dan penggantian (*replacement*).

Edit distance dihitung dengan menggunakan matriks *Dynamic Programming*. Solusi didapat dengan mengkomputasikan matriks $C[0..m,0..n]$, dimana $C[i,j]$ adalah nilai perbedaan suatu substring dengan polanya, dengan relasi rekursif sebagai berikut:

$$C[i,j] = \min \begin{cases} C[i-1,j-1] + \delta(p_i,t_j) \\ C[i,j-1] + c \\ C[i-1,j] + c \end{cases} \quad (1)$$

dengan nilai awal $C[0,j] = 0$ dan $C[i,0] = ic$ serta nilai *edit distance* yang sering dipakai adalah $c = 1$ dan $\delta(a,b) = 1$, jika $a \neq b$. Nilai $C[m,j]$ memberikan nilai minimal perbedaan antara suatu substring dengan polanya. Nilai $C[m,j]$ biasa disebut dengan *score*. Komputasi dikerjakan dengan menggunakan matriks, dimana setiap elemennya merupakan nilai untuk $C[i,j]$.

Contoh untuk permasalahan memperkirakan kecocokan string dengan algoritma *edit distance* adalah:

Diketahui teks T = bacabbbbac dan pola P = bbba. Dari teks tersebut, akan dicari substring mana saja yang diperkirakan mirip dengan pola dengan maksimal perbedaan dua.

Dibentuk tabel untuk nilai $C[i,j]$ (yang sering disebut sebagai matriks DP (*Dynamic Programming*)) seperti ditunjukkan pada Tabel 1.

Tabel 1 Tabel *Dynamic Programming* untuk C[*i*,*j*]

			TEKS									
			b	a	c	a	b	b	b	b	a	c
Posisi			1	2	3	4	5	6	7	8	9	10
		0	0	0	0	0	0	0	0	0	0	0
P	b	1	0	1	1	1	0	0	0	0	1	1
O	b	2	1	1	2	2	1	0	0	0	1	2
L	b	3	2	2	2	3	2	1	0	0	1	2
A	a	4	3	2	3	2	3	2	1	1	0	1

Untuk menentukan substring-substring yang diperkirakan cocok dengan maksimal perbedaan tertentu, ditentukan pola sebagai berikut:

1. Jika $C[m,j] = C[m-1,j-1] + \delta(p_i,t_j)$, maka panjang substring yang diperkirakan cocok kurang dari atau sama dengan panjang pola. Karakter pertama dari substring yang diperkirakan cocok diambil dari karakter-karakter sebelumnya sepanjang panjang pola, Jika panjang substring yang diperkirakan cocok ternyata kurang dari panjang pola, maka karakter pertama diambil dari karakter pertama teks. Karakter terakhir merupakan karakter pada posisi j.
2. Jika $C[m,j] = C[m,j-1] + c$, maka panjang substring yang diperkirakan cocok lebih dari panjang pola. Karakter pertama dari substring yang diperkirakan

cocok diambil dari karakter yang mempunyai *score* sama dengan maksimal perbedaan. Karakter terakhir adalah karakter pada posisi *j*.

3. Jika $C[m,j] = C[i\text{-}1,j] + c$, maka panjang substring yang diperkirakan cocok kurang dari panjang pola. Untuk mencari substring yang diperkirakan cocok, terlebih dahulu dicari elemen $C[i,j] = 0$, $i = 1, \ldots,$ panjang pola. Dari posisi tersebut akan didapat karakter pertama dari substring yang diperkirakan cocok dengan melacak secara diagonal sampai pada posisi $i = 1$. Karakter terakhir adalah karakter pada posisi *j*.

Dari hasil pada Tabel 1, terlihat bahwa jika maksimal perbedaan dua, maka substring yang diperkirakan cocok adalah:

Untuk maksimal perbedaan 0 (substring sama dengan pola), terjadi pada substring: “bbba” (posisi 6-9); Untuk maksimal perbedaan 1, terjadi pada substring: “bbb” (posisi 5-7) yaitu “a” pada posisi 4 dari pola dihilangkan, “bbbb” (posisi 5-8) yaitu “a” pada posisi 4 dari pola diganti dengan “b” pada teks, “bbbac” (posisi 6-10) yaitu “c” pada posisi 10 dari teks disisipkan pada posisi 5 pada pola; Untuk maksimal perbedaan 2, terjadi pada substring: “ba” (posisi 1-2) yaitu “bb” pada posisi 2 dan 3 dari pola dihilangkan, “baca” (posisi 1-4) yaitu “bb” pada posisi 2 dan 3 dari pola diganti dengan “ac” pada teks, “bb” (posisi 5-6) yaitu “ba” pada posisi 3 dan 4 dari pola dihilangkan.

4. Vektorisasi

Istilah vektorisasi mengacu pada pengurangan kode-kode program yang menggunakan loop *for* atau *while* setiap skalar dalam suatu *array*. Perhitungan dapat berjalan secara paralel. Untuk melakukan vektorisasi suatu kode sangat tergantung pada masalah yang dihadapi. Kode-kode yang tervektorisasi selalu lebih cepat dieksekusi daripada kode-kode yang tidak tervektorisasi.

5. Algoritma Vektor untuk Perkiraan Kecocokan String

Permasalahan memperkirakan kecocokan string dapat dipercepat dengan memanfaatkan keparalelan vektor pada string dengan algoritma vektor bit. Jika diketahui $\mathbf{x} = x_1\, x_2 \ldots x_m$ dan $\mathbf{y} = y_1\, y_2 \ldots y_m$, adalah vektor boolean/vektor bit, maka notasi yang digunakan untuk permasalahan memperkirakan kecocokan string adalah: NOT **x** (~ **x**) , **x** OR **y** (**x** | **y**), **x** AND **y** (**x** & **y**), **x** XOR **y** (**x** $\wedge$ **y**), **x** ADD **y** (**x**+**y**), SHL **x** (**x** >> *a*), SHR **x** (**x** << *a*), dengan *a* adalah suatu konstanta.

Jika didefinisikan:

$$\Delta h_{i,j} = C[i,j] - C[i,j\text{-}1] \in \{-1, 0, +1\} \quad (2)$$

$$\Delta v_{i,j} = C[i,j] - C[i\text{-}1,j] \in \{-1, 0, +1\} \quad (3)$$

$$\Delta d_{i,j} = C[i,j] - C[i\text{-}1,j\text{-}1] \in \{0, +1\} \quad (4)$$

dan variabel boolean:

$$VP_{i,j} \equiv (\mathrm{D}v_{i,j} = +1) \quad (5)$$
$$VN_{i,j} \equiv (\mathrm{D}v_{i,j} = -1) \quad (6)$$
$$HP_{i,j} \equiv (\mathrm{D}h_{i,j} = +1) \quad (7)$$
$$HN_{i,j} \equiv (\mathrm{D}v_{i,j} = -1) \quad (8)$$
$$D0_{i,j} \equiv (\mathrm{D}d_{i,j} = 0) \quad (9)$$

maka didapat rumus sebagai berikut:

$$HN_{i,j} \Leftrightarrow VP_{i,j-1} \text{ AND } D0_{i,j} \quad (10)$$
$$VN_{i,j} \Leftrightarrow HP_{i-1,j} \text{ AND } D0_{i,j} \quad (11)$$
$$HP_{i,j} \Leftrightarrow VN_{i,j-1} \text{ OR NOT } (VP_{i,j-1} \text{ OR } D0_{i,j}) \quad (12)$$
$$VP_{i,j} \Leftrightarrow HN_{i-1,j} \text{ OR NOT } (HP_{i-1,j} \text{ OR } D0_{i,j}) \quad (13)$$
$$D0_{i,j} \Leftrightarrow (p_i = t_j) \text{ OR } VN_{i,j-1} \text{ OR } HN_{i-1,j} \quad (14)$$

Algoritma vektor bit untuk permasalahan memperkirakan kecocokan string adalah sebagai berikut:

Inisialisasi:

```
for c ∈ a do
B[c] = 0^m
end for
for j ∈ 1 .. m do
   B[p_j] = B[p_j] | 0^(m-j) 1 0^(j-1)
end for
VP = 1^m
VN = 0^m
score = m
```

Proses Utama:

```
for pos ∈ 1 .. n do
   X = B[t_pos] | VN
   D0 = ((VP + (X&VP)) ∧ VP) | X
HN = VP & D0
   HP = VN | ~ (VP | D0)
   X = HP << 1
   VN = X & D0
   VP = (HN << 1) | ~ (X | D0)
   if HP & 10^(m-1) ¹ 0^m
        then score+ = 1
   else if HN & 10^(m-1) ¹ 0^m
        then score– = 1
   else if HP & 10^(m-1) = 0^m
        then score = score
   end if
   if score £ k
       then print(pos)
   end if
end for
```

Contoh untuk permasalahan memperkirakan kecocokan string dengan

algoritma vektor bit adalah:

Diketahui teks *T* = bacabbbbac dan pola *P* = bbba, akan dicari pada posisi substring manakah pola tersebut diperkirakan mirip, dengan maksimal perbedaan dua.

Pola *P* = "bbba" → terdiri dari alfabet a = {a, b}.

Dibentuk vektor pola B(b) dan B(a), yaitu untuk setiap posisi *j*, dengan *j* = 1, …, *m*, dengan B(p_j) = 1, jika $p_j = a_i$ dan B(p_j) = 0, jika $p_j \neq a_i$, *i* adalah banyak alfabet.

"b"= 0 0 0 1
"a" = 1 0 0 0

Dalam perhitungannya

Inisialisasi:

VN = 0 0 0 0
VP = 1 1 1 1
Score = 4

Proses Utama:

Dibaca karakter pertama dari teks yaitu "b"

"b" = 0 0 0 1
*D*0 = 1 1 1 1
HN = 1 1 1 1
HP = 0 0 0 0
VN = 0 0 0 0
VP = 1 1 1 0
Score = 3

Dibaca karakter kedua dari teks yaitu "a"

"a" = 1 0 0 0
*D*0 = 1 0 0 0
HN = 1 0 0 0
HP = 0 0 0 1
VN = 0 0 0 0
VP = 0 1 0 1
Score = 2
Posisi = 2

Dibaca karakter ketiga dari teks yaitu "c"

"c" = 0 0 0 0
*D*0 = 0 0 0 0
HN = 0 0 0 0
HP = 1 0 1 0
VN = 0 0 0 0
VP = 1 0 1 1
Score = 3

Dibaca karakter keempat dari teks yaitu "a"

"a" = 1 0 0 0
*D*0 = 1 0 0 0
HN = 1 0 0 0
HP = 0 1 0 0
VN = 1 0 0 0
VP = 0 1 1 1
Score = 2
Posisi = 4

Dibaca karakter kelima dari teks yaitu "b"

"b" = 0 0 0 1
*D*0 = 1 1 1 1
HN = 0 1 1 1
HP = 1 0 0 0
VN = 0 0 0 0
VP = 1 1 1 0
Score = 3

Dibaca karakter keenam dari teks yaitu "b"

"b" = 0 0 0 1
*D*0 = 1 1 1 1
HN = 1 1 1 0
HP = 0 0 0 0
VN = 0 0 0 0
VP = 1 1 0 0
Score = 2
Posisi = 6

Dibaca karakter ketujuh dari teks yaitu "b"

"b" = 0 0 0 1
*D*0 = 1 1 1 1
HN = 1 1 0 0
HP = 0 0 0 0
VN = 0 0 0 0
VP = 1 0 0 0
Score = 1
Posisi = 7

Dibaca karakter kedelapan dari teks yaitu "b"

"b" = 0 0 0 1
*D*0 = 0 1 1 1
HN = 0 0 0 0
HP = 0 0 0 0
VN = 0 0 0 0
VP = 1 0 0 0

Score = 1
Posisi = 8

Dibaca karakter kesembilan dari teks yaitu “a”

“a” = 1 0 0 0
*D*0 = 1 0 0 0
HN = 1 0 0 0
HP = 0 1 1 1
VN = 1 0 0 0
VP = 0 0 0 1
Score = 0
Posisi = 9

Dibaca karakter kesepuluh dari teks yaitu “c”

“c” = 0 0 0 0
*D*0 = 1 0 0 0
HN = 0 0 0 0
HP = 1 1 1 0
VN = 1 0 0 0
VP = 0 0 1 1
Score = 1
Posisi = 10

Dari perhitungan di atas, diperoleh hasil perkiraan kecocokan string seperti tampak pada Tabel 2.

Tabel 2 Hasil Perkiraan Kecocokan String dengan Algoritma Vektor Bit

POLA	b	b	b	a					
TEKS	b	a	c	a	b	b	b	b	a	c
Posisi	1	2	3	4	5	6	7	8	9	10
Score	3	**2**	3	**2**	3	**2**	**1**	**1**	**0**	**1**

Dengan ketentuan pola yang sama seperti pada Tabel 2, maka substring yang diperkirakan mirip dengan pola “bbba” dengan maksimal perbedaan dua adalah:

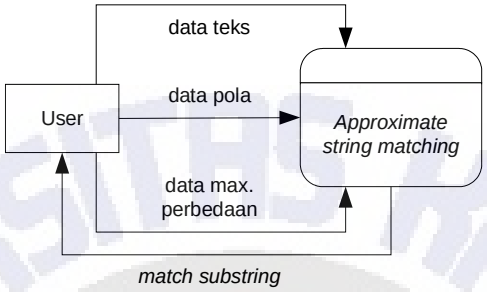
“ba” (posisi 1-2), “baca” (posisi 1-4), “abb” (posisi 4-6), “bbb” (posisi 5-7), “bbbb” (posisi 4-7), “bbba” (posisi 6-9), dan “bbbac” (posisi 6-10).

Untuk merancang suatu sistem dibutuhkan tahap analisis yang akan dibahas sebagai berikut.

Data flow diagram (DFD)/diagram alir data (DAD) adalah sebuah teknik grafis yang menggambarkan aliran informasi dan transformasi yang diaplikasikan pada saat data bergerak dari input menjadi output.

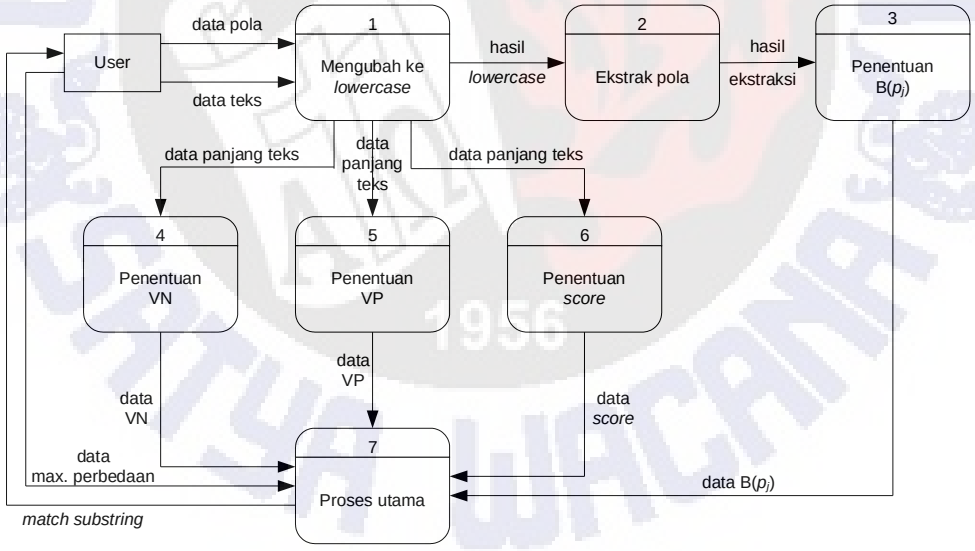
DFD digunakan untuk menyajikan sebuah sistem atau perangkat lunak

pada setiap tingkatan abstraksi. DFD dapat dipartisi ke dalam tingkat-tingkat yang merepresentasikan aliran informasi yang bertambah. DFD memberikan suatu mekanisme bagi pemodelan fungsional dan pemodelan aliran informasi. DFD untuk permasalahan memperkirakan kecocokan string (*approximate string matching*) ditampilkan pada Gambar 1.



Gambar 1 Diagram Konteks

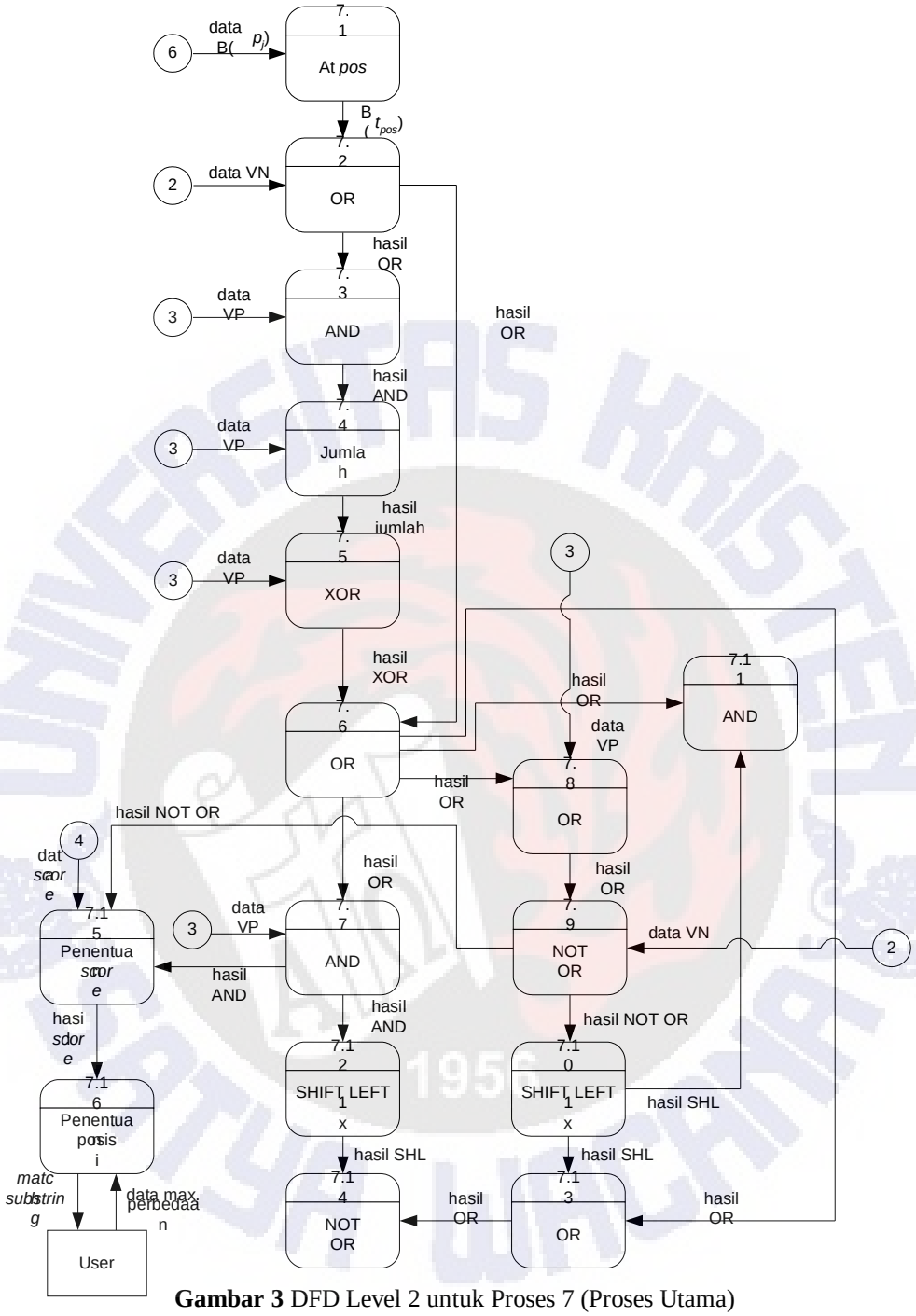
Proses *approximate string matching* dapat dipecah menjadi beberapa proses, seperti yang ditampilkan pada Gambar 2.



Gambar 2 DFD Level 1

Untuk proses utama, dibagi menjadi beberapa proses seperti yang ditampilkan pada Gambar 3.

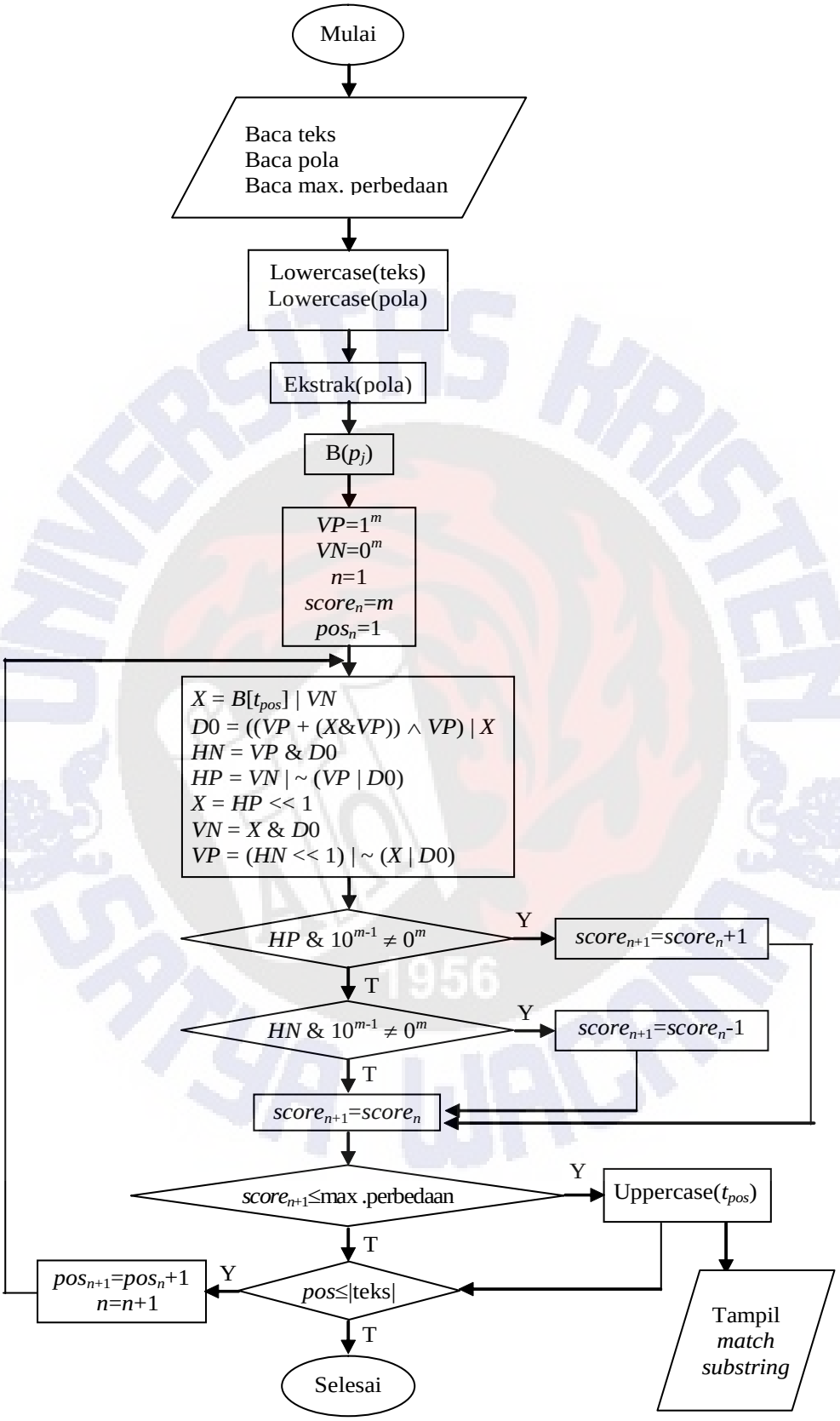
Selanjutnya dapat dibuat *flowchart* untuk permasalahan memperkirakan kecocokan string (*approximate string matching*) sebagai yang tertampil pada Gambar 4.



Gambar 3 DFD Level 2 untuk Proses 7 (Proses Utama)

6. Aplikasi Algoritma Vektor pada Uji Kecocokan DNA (*DNA Matching*)

Permasalahan kecocokan string (*approximate string matching*) dapat diaplikasikan pada bidang biologi, yaitu untuk uji kecocokan DNA (*DNA matching*).



Gambar 4 *Flowchart* Algoritma Bit *Vector*

DNA (*Deoxyribo Nucleic Acid* = Asam Deoksiribosa Nukleat) terdapat dalam semua sel yang sedang mengalami pembelahan diri, dimana sebagian besar DNA terdapat di dalam nukleus (inti sel) dan merupakan komponen kromosom yang sangat penting. Banyak data yang mengungkapkan bahwa DNA adalah pembawa sebagian besar atau seluruh sifat-sifat genetik yang khas dalam kromosom, sehingga DNA merupakan gen itu sendiri.

DNA merupakan susunan kimia makro molekuler yang kompleks. Moluekulnya merupakan suatu rantai yang amat panjang dan substansi dasarnya terdiri atas tiga yaitu Gula pentosa (deoksiribosa), Fosfat (PO_4^-), Basa nitrogen, yang dapat dibedakan menjadi empat macam yang dikelompokkan dalam dua kelompok yaitu golongan pirimidin, yang terdiri dari Timin (T) dan Cytosine/ Sitosin (C/S) dan golongan purin, yang terdiri dari Guanin (G) dan Adenin (A) [5].

Permasalahan memperkirakan kecocokan string ini dapat diaplikasikan pada bidang biologi, yaitu untuk uji kecocokan DNA (*DNA Matching*), yang digunakan untuk mengidentifikasi dan membandingkan pola DNA dari suatu sampel untuk dicocokkan dengan pola DNA yang lain berdasarkan maksimal perbedaan yang telah ditentukan.

Software yang digunakan untuk permasalahan memperkirakan kecocokan string (*approximate string matching*) ini adalah MATLAB. Penulis menggunakan *software* MATLAB dikarenakan MATLAB mampu bekerja dengan konsep vektor, mampu bekerja dengan konsep matriks, dan mempunyai perintah-perintah khusus dalam hubungannya dengan operator logika *bit-wise*.

Dari contoh yang sudah dibahas di atas, input data yang dimasukkan dalam MATLAB adalah data teks, data pola, dan data maksimal perbedaan.

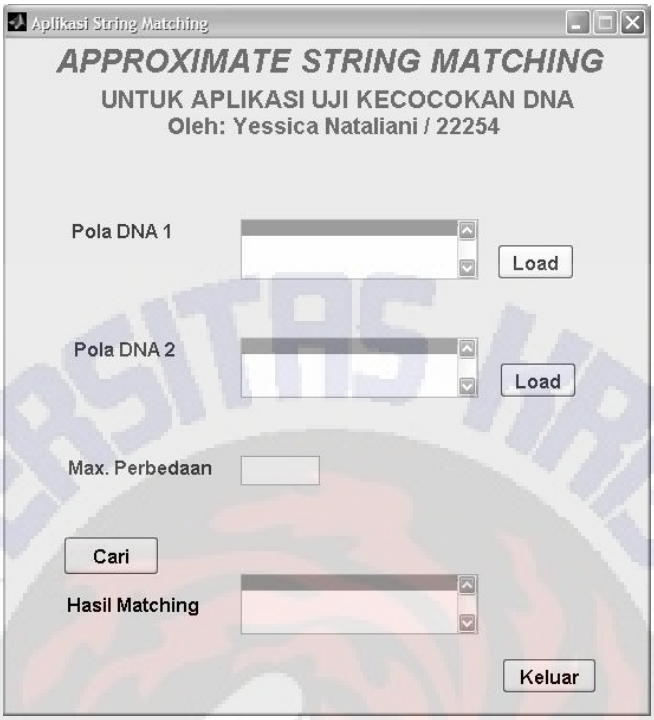
Data teks : bacabbbbac
Data pola : bbba
Data maksimal perbedaan : 2

Hasil yang tertampil pada MATLAB adalah “bAcABBBBAC”, yang berarti substring yang mempunyai karakter terakhir pada posisi 2, 4, 6, 7, 8, 9, 10 (posisi karakter huruf besar) diperkirakan mirip dengan pola yang telah diinputkan sebelumnya, dengan perbedaan tidak lebih dari dua, yaitu “ba” (posisi 1-2), “baca” (posisi 1-4), “abb” (posisi 4-6), “bbb” (posisi 5-7), “bbbb” (posisi 4-8), “bbba” (posisi 6-9), “bbbac” (posisi 6-10).

Sedangkan untuk aplikasi dalam bidang biologi, yaitu untuk uji kecocokan DNA yang menggunakan sampel DNA burung yang diambil dari dari DDBJ Database. Sebagai contoh, diambil pola DNA pertama dari spesies *Aepypodius arfakianus* yaitu ctgggtggtgagtttcccaaattttcaaacgtttgccagaaatttc cctaga dan pola DNA kedua dari spesies *Eulipoa wallacei* yaitu ctgggtggtgagttt cccaaattttcaaatttttaggaattttccctagatt dengan maksimal perbedaan sembilan.

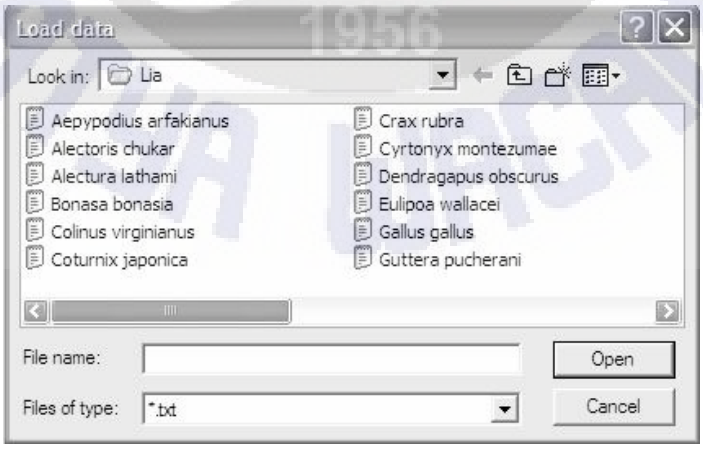
Panjang pola DNA 1 dan pola DNA 2 diambil 52 karakter. Dari 52 karakter tersebut, diambil maksimal 10 perbedaan basa (± 20%), supaya kedua pola DNA diperkirakan mirip.

Tampilan awal menu yang dibuat sebagai aplikasi untuk uji kecocokan DNA seperti yang tertampil pada Gambar 5.



Gambar 5 Tampilan Awal Menu untuk Aplikasi

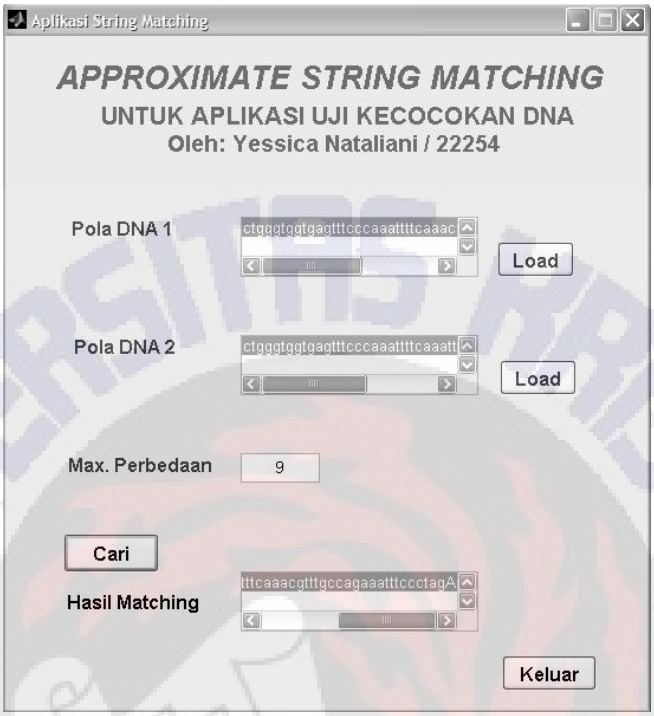
Pola DNA diambil dari database yang sudah dibuat. Jika ditekan tombol Load, akan tertampil data-data DNA dari beberapa spesies (dalam hal ini famili Aves pada gen Rhodopsin (RDP1)) yang sudah tersimpan dalam *file* teks seperti yang tertampil pada Gambar 6.



Gambar 6 Tampilan *Load* Data

Sebagai contoh akan diperkirakan kecocokan antara DNA spesies *Aepypodius arfakianus* dengan *Eulipoa wallacei* seperti pada contoh yang

sudah ditulis sebelumnya. Hasil *approximate string matching* untuk aplikasi uji kecocokan DNA Aves adalah seperti yang tertampil pada Gambar 7.



Gambar 7 Tampilan Akhir Menu untuk Aplikasi

Hasil yang tertampil dari proses yang telah dilakukan adalah "ctgggtggtgagtttcccaaattttcaaacgtttgccagaaatttccctagA". Dengan menginputkan maksimal perbedaan sembilan, ternyata huruf terakhir (karakter pada hasil yang tertampil merupakan pola DNA dari pola DNA 1) menjadi huruf besar. Hal ini menunjukkan bahwa antara pola DNA *Aepypodius arfakianus* dengan pola DNA *Eulipoa wallacei* paling sedikit ada sembilan perbedaan basa, supaya keduanya diperkirakan mirip.

7. Simpulan

Dari pembahasan permasalahan memperkirakan kecocokan string (*approximate string matching*) serta aplikasinya dalam bidang biologi, yaitu untuk uji kecocokan DNA (*Deoxyribo Nucleid Acid*), maka dapat diambil beberapa simpulan sebagai berikut: (1) Permasalahan memperkirakan kecocokan string (*approximate string matching*) yang biasanya menggunakan perhitungan *edit distance*, berdasarkan jumlah operasi yang dibutuhkan yang terdiri dari penghapusan, penyisipan, dan penggantian untuk mengubah suatu string ke string yang lain, ternyata dapat dipercepat dengan proses vektorisasi yang mengurangi proses pengulangan dengan memanfaatkan keparalelan

vektor pada string dan (2) Permasalahan memperkirakan kecocokan string (*approximate string matching*) dapat diaplikasikan pada bidang yang lain, seperti bidang biologi, misalnya untuk uji kecocokan DNA yang dapat digunakan untuk mengidentifikasi dan membandingkan pola DNA dari suatu sampel untuk dicocokkan dengan pola DNA yang lain.

8. Daftar Pustaka

[1] Ullman, J. D., Hopcroft J., 2005, *Introduction to Automata Theory Language and Computation*, Addison Wesley, Canada.

[2] Bergeron, A. dan Hamel, S., *Fast Implementation of Automata Computation*, Lacim, Canada, (http://www.labmath.ugam.ca/~hamel/fast.pdf). Diakses tanggal 16 Februari 2006.

[3] Bergeron, A. dan Hamel, S., *Vector Algorithms for Approximate String Matching*, Lacim, Canada, (http://www.lacim.ugam.ca/~hamel/IJFCS.pdf). Diakses tanggal 20 Februari 2006.

[4] Huson, D., 2004, *Bit-Vector-Based Approximate String Matching*, (http://www.inf. fu-berlin.de/inst/ag-bio/FILES/ROOT/Teaching/Lectures/WS0405//script%20 seqanI/script_2_approximate_searching.pdf). Diakses tanggal 20 Februari 2006.

[5] Pratiwi, D.A., Maryati, S., Srikini, Suharno, dan Bambang, 1997, *Biologi SMU Kelas 3 Jilid 3 Kurikulum 1994*, Erlangga, Jakarta.

[6] Cronkite, Donald, 2002, *Cell and Heredity*, Prentice-Hall, Inc., New Jersey.

[7] *DDBJ Database RDP1*, (http://www.ddbj.nig.ac.jp/cgi-bin/wgetz). Diakses tanggal 24 Agustus 2006.