BAB 4

IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1 Implementasi Sistem

Tahap implementasi sistem adalah tahap pembuatan perangkat lunak yang merupakan tahap lanjut dari tahap perancangan sistem. Tahap ini dilakukan untuk menerjemahkan perancangan berdasarkan hasil analisis dalam bahasa yang dimengerti oleh komputer serta penerapan perangkat lunak pada keadaan yang sebenarnya.

4.1.1 Implementasi Perangkat Keras

Spesifikasi perangkat keras (*hardware*) yang digunakan untuk membangun aplikasi perbandingan Algoritma *Reverse Colussi* dan *Zhu Takaoka* ini dapat dilihat pada *Tabel 4.1 Spesifikasi Perangkat Keras yang Digunakan*.

Tabel 4.1 Spesifikasi Perangkat Keras yang Digunakan

No	Perangkat Keras	Spesifikasi
1	Prosessor	Core i3 2.53 GHz
2	Monitor	14”, resolusi 1366 x 768
3	VGA	Intel On Board
4	Memory	4096MB DDR3
5	Keyboard dan Mouse	Standard, Port USB

4.1.2 Implementasi Perangkat Lunak

Spesifikasi perangkat lunak (*software*) yang digunakan untuk membangun aplikasi perbandingan Algoritma *Reverse Colussi* dan *Zhu Takaoka* ini dapat dilihat pada *Tabel 4.2 Spesifikasi Perangkat Lunak yang Digunakan*.

Tabel 4.2 Spesifikasi Perangkat Lunak yang Digunakan

No	Perangkat Lunak	Spesifikasi
1	Sistem Operasi	Windows 7 Ultimate 32 bit, Ubuntu 16.04 64bit
2	Tools Pembangun Perangkat Lunak	Virtual box, Sublime 3.0, Dev C++, Terminal, Qt Creator, Notepad ++, gedit.
3	Tools Pemodelan Sistem	Microsoft Visio

4.1.3 Implementasi Antarmuka

Implementasi antarmuka dilakukan untuk mengimplementasikan setiap tampilan yang terdapat pada tahap perancangan yang telah dilakukan. Adapun implementasi antarmuka dari aplikasi yang telah dibuat dapat dilihat pada *Tabel 4.3 Implementasi Antarmuka*, di bawah ini.

Tabel 4.3 Implementasi Antarmuka

No	Form Antarmuka	Proses	Nama File	Keterangan
1	Form menu utama	Menampilkan menu utama.	mainwindow.ui	Tampilan utama untuk menampilkan semua fitur yang terdapat pada aplikasi
2	Form pilih folder master	Mengatur posisi folder utama yang berisi inti dari sistem.	setupwindow.ui	Tampilan untuk mencari folder utama dari program.
3	Form tentang	Menampilkan informasi tentang pembuat program.	aboutwindow.ui	Tampilan untuk memberikan informasi tentang profil pembuat aplikasi

Gambar implementasi antarmuka dapat dilihat pada bagian Lampiran A.

4.2 Pengujian Sistem

Pengujian sistem merupakan bagian yang penting dalam pembangunan sebuah perangkat lunak. Pengujian ditujukan untuk menemukan kesalahan-kesalahan pada sistem dan memastikan sistem yang dibangun telah sesuai dengan apa yang direncanakan sebelumnya. Pengujian juga dilakukan untuk menjamin kualitas dan mengetahui kelemahan dari perangkat lunak. Tujuan dari pengujian ini adalah untuk menjamin bahwa perangkat lunak yang dibangun memiliki kualitas yang handal, yaitu mampu mempresentasikan kajian pokok dari spesifikasi analisis, perancangan dari perangkat lunak itu sendiri.

4.2.1 Rencana Pengujian

Rancangan pengujian yang akan dilakukan dalam pembangunan sistem perbandingan algoritma *Reverse Colussi* dan *Zhu Takaoka* dibagi menjadi 2 bagian yaitu teknik pengujian *black box* dan teknik pengujian *white box*. Pengujian *black box* berfokus pada pengujian semua perintah-perintah dan kondisi yang di eksekusi oleh perangkat lunak dan pengujian ini dilakukan oleh pembangun aplikasi. Pengujian *white box* dilakukan untuk menguji *function* pada sistem dan menguji apakah *function* tersebut memenuhi kebutuhan sistem.

Tabel 4.4 Rencana Pengujian Fungsional

No	Kelas Uji	Butir Pengujian	Jenis Pengujian
1	Mencocokan Pola (Data pada sistem)	Pencocokan pola sumber *string* dengan *pattern*.	*Black box*
2	Mencocokan Pola (Data masukan *user*)	Pencocokan pola sumber *string* dengan *pattern*.	*Black box*
3	Uji Algoritma	Pencocokkan algoritma *Reverse Colussi*	*White box*
		Pencocokkan algoritma *Zhu-Takaoka*	*White box*

4.2.1.1 Kasus dan Hasil Pengujian *Black Box*

Pengujian *black box* dilakukan dengan tujuan untuk mengetahui apakah data yang dimasukan sudah benar atau belum, hasil *output* apakah sesuai dengan harapan atau tidak. Berikut adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *black box* berdasarkan pengujian yang sudah dilakukan dan menghasilkan kesimpulan dari pengujian tersebut.

Tabel 4.5 Pengujian Mencocokan Pola (Data pada sistem)

Kasus dan Hasil Uji (Data Benar)			
Data Masukan	**Hasil yang diharapkan**	**Pengamatan**	**Kesimpulan**
Sumber *string*: Pasukan Haji Prawatasari (Bagaian 2).txt *Pattern*: Sunjaya	Menampilkan waktu pencarian yang digunakan oleh kedua algoritma.	Sistem menampilkan waktu *preprocessing* dan pencocokan pola yang digunakan oleh kedua algoritma.	[√] Berhasil [ ] Tidak Berhasil
Kasus dan Hasil Uji (Data Salah)			
Sumber *string*: asdf.doc *Pattern*: Sunjaya	Tidak dapat melakukan pencocokan.	Sistem menampilkan pesan tidak dapat melakukan pencocokan.	[√] Berhasil [ ] Tidak Berhasil

Tabel 4.6 Pengujian Mencocokan Pola (Data masukan *user*)

Kasus dan Hasil Uji (Data Benar)			
Data Masukan	**Hasil yang diharapkan**	**Pengamatan**	**Kesimpulan**
Sumber *string*: asdfgalindaalindaalindaalindasangkilar *Pattern*: linda	Menampilkan waktu pencarian yang digunakan oleh kedua algoritma.	Sistem menampilkan waktu *preprocessing* dan pencocokan pola yang digunakan oleh kedua algoritma.	[√] Berhasil [ ] Tidak Berhasil
Kasus dan Hasil Uji (Data Salah)			
Sumber *string*: alkaa akalak adkfl *Pattern*: kala	Tidak dapat melakukan pencocokan.	Sistem menampilkan pesan tidak dapat melakukan pencocokan.	[√] Berhasil [ ] Tidak Berhasil

4.2.1.1.1 Kesimpulan Hasil Pengujian *Black box*

Berdasarkan hasil pengujian *black box* (fungsional) dengan kasus uji di atas, dapat ditarik kesimpulan bahwa pembangunan aplikasi perbandingan Algoritma *Reverse Colussi* dan *Zhu Takaoka* tidak terdapat kesalahan. Pada proses secara fungsional aplikasi yang dibangun mengeluarkan hasil yang sesuai dengan harapan.

4.2.1.2 Kasus dan Hasil Pengujian *White Box*

Pengujian *white box* bertujuan untuk mengetahui kinerja logika yang dibuat pada sebuah perangkat lunak, apakah sudah berjalan dengan baik atau tidak. Pengujian *white box* akan digunakan pada algoritma *Zhu-Takaoka* dan *Reverse Colussi* untuk mengukur kinerja logika berdasarkan *pseudocode* yang telah dibuat pada tahap analisis. Pengujian *white box* dilakukan dengan beberapa tahap, yaitu :

1. Langakah pertama ubah *source code* menjadi *flowgraph*.
2. Tahap pengujian, dimana tahap pengujian ini dilakukan dengan 5 cara yaitu menghitung *region*, menghitung *cyclomatic complexity*, menghitung *independent path*, menggunakan *graph matriks* dan menghitung *predicate node*.

4.2.1.2.1 Pengujian *White Box* algoritma *Reverse Colussi*

Berikut ini adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *White Box* dari algoritma *Reverse Colussi* pada fase pencocokkan *pattern* berdasarkan *source code* algoritmanya dan menghasilkan kesimpulan pengujian tersebut.

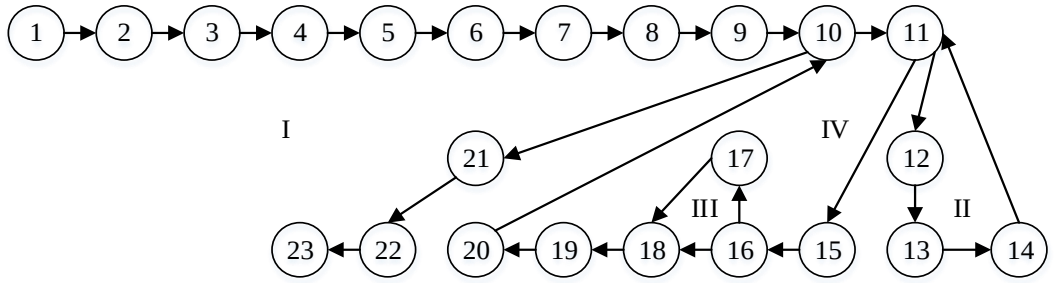
Tabel 4.7 *Source Code* Algoritma *Reverse Colussi*

No	Reverse Colussi (Pencocokan *Pattern*)
1	`int search(unsigned char *x, int m, unsigned char *y, int n) {`
2	`int i, j, s, rcBc[SIGMA][XSIZE], rcGs[XSIZE], h[XSIZE], count;`
	`/* Preprocessing */`
3	`BEGIN_PREPROCESSING`
4	`preRc(x, m, h, rcBc, rcGs);`
5	`END_PREPROCESSING`
	`/* Searching */`
6	`BEGIN_SEARCHING`
7	`count = 0;`
8	`j = 0;`
9	`s = m;`
10	`while (j <= n - m) {`

```
11          while (j <= n - m && x[m - 1] != y[j + m - 1]) {
12             s = rcBc[y[j + m - 1]][s];
13             j += s;
14          }
15          for (i = 1; i < m && x[h[i]] == y[j + h[i]]; ++i);
16          if (i >= m && j<=n-m)
17             OUTPUT(j);
18          s = rcGs[i];
19          j += s;
20       }
21       END_SEARCHING
22       return count;
23    }
```

1. Mengubah *source code* menjadi *Flow Graph*

Gambar 4.1 Flow Graph Algoritma Reverse Colussi ini menunjukan *flow graph* algoritma *Reverse Colussi* dalam melakukan fase pencocokkan *string* berdasarkan *pseudocode*.



Gambar 4.1 *Flow Graph* Algoritma *Reverse Colussi*

Keterangan :

Node = Menggambarkan Kondisi

Edge = Menggambarkan Aksi

2. Tahap Pengujian

a) *Region*

Region = 4

b) *Cylomatic Complexity V(G)*

Dari *flow graph* di atas, dapat dihitung *Cylomatic Complexity*, yakni *Region* = 4 yang dapat dari perhitungan dibawah ini.

$V_{(g)}$ = Edge – Node + 2

$V_{(g)}$ = 25 - 23 + 2 = 4

c) *Independent Path*

*Path*1 = 1-2-3-4-5-6-7-8-9-10-21-22-23

*Path*2 = 1-2-3-4-5-6-7-8-9-10-11-15-16-17-18-19-20-10-21-22-23

*Path*3 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-11-15-16-18-19-20-10-21-22-23

*Path*4 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-11-15-16-17-18-19-20-10-21-22-23

d) *Predicate Node*

$V_{(G)}$ = jumlah node yang memiliki lebih dari satu jalur + 1

$V_{(G)}$ = 3 + 1 = 4

e) *Graph Matric*

Penomoran ulang *flowgraph* atau penyederhanaan berdasarkan kondisi atau simpul dapat dilihat pada *Gambar 4.2 Graph Matric Algoritma Reverse Colussi.*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	N(E)-1
1		1																						0
2			1																					0
3				1																				0
4					1																			0
5						1																		0
6							1																	0
7								1																0
8									1															0
9										1														0
10											1										1			1
11												1			1									1
12													1											0
13														1										0
14											1													0
15																1								0
16																	1	1						1
17																		1						0
18																			1					0
19																				1				0
20										1														0
21																					1			0
22																						1		0
Sum(E) + 1																								4

Gambar 4.2 *Graph Matric Algoritma Reverse Colussi*

Tabel 4.8 Uji Coba *Independ Path*

***Independent Path* ke**	**Penggunaan**	**Contoh data masukan**	**Yang diharapkan**	**Hasil**
1	Ketika perdeicate node 1 bernilai false	Nyi Rodiah	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
2	Ketika perdeicate node 1 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
3	Ketika perdeicate node 2 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
4	Ketika perdeicate node 3 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil

Langkah terakhir dalam melakukan pengujian menggunakan metode *basis path testing* adalah menyiapkan kasus-kasus uji untuk mengeksekusi semua alur logika yang telah dibuat pada langkah sebelumnya. Kasus uji yang dimaksud disini adalah dengan cara memberikan nilai pada variabel yang terlibat. Nilai yang dimasukan haruslah nilai yang mungkin muncul dan sesuai dengan tipe data yang telah didefinisikan. *Test case* dibuat dalam bentuk tabel yang tujuannya untuk mempermudah eksekusi setiap *basis path*, karena nilai-nilai yang dimasukan dan yang dihasilkan akan lebih mudah diamati.

1) Pengujian *Path* 1 = 1-2-3-4-5-6-7-8-9-10-21-22-23

   Pada *path* ini, supaya *node* 10 dapat menuju ke *node* 21, maka nilai dari *node* 10 harus dapat menghasilkan nilai *false*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya tidak terdapat pola *Nyi Rodiah*. Agar bernilai *false* digunakan pola *Nyi Rodiah* sebagai data uji coba pencocokan pola.

2) Pengujian *Path* 2 = 1-2-3-4-5-6-7-8-9-10-11-15-16-17-18-19-20-10-21-22-23

   Pada *path* ini, supaya *node* 10 dapat menuju ke *node* 11, maka nilai dari *node* 10 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan

didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

3) Pengujian *Path* 3 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-11-15-16-18-19-20-10-21-22-23

   Pada *path* ini, supaya *node* 11 dapat menuju ke *node* 15, maka nilai dari *node* 11 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

4) Pengujian *Path* 4 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-11-15-16-17-18-19-20-10-21-22-23

   Pada *path* ini, supaya *node* 16 dapat menuju ke *node* 17, maka nilai dari *node* 16 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

3. Kesimpulan

Berdasarkan pengujian pada setiap metode, dihasilkan nilai *Cyclomatic Complexity* yang sama yaitu 4. Maka dapat disimpulkan bahwa pengujian *white box* pada algoritma *Reverse Colussi* berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

4.2.1.2.2 Pengujian *White Box* algoritma *Zhu-Takaoka*

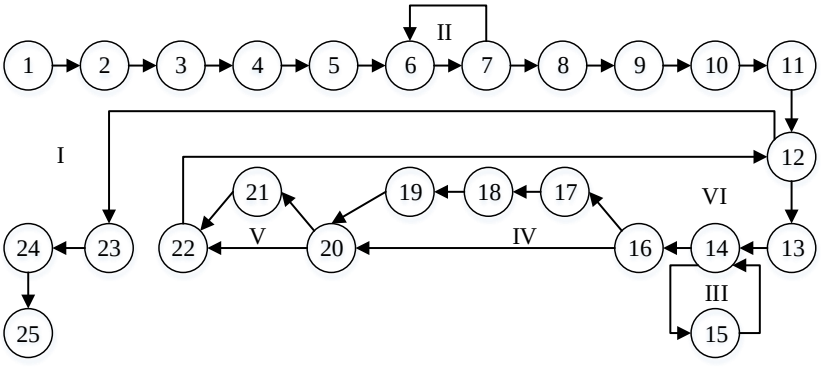
Berikut ini adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *White Box* dari algoritma *Zhu Takaoka* pada pencocokkan *pattern* berdasarkan *source code* algoritmanya dan menghasilkan kesimpulan pengujian tersebut.

Tabel 4.9 *Source Code* Algoritma *Zhu Takaoka*

No	Zhu Takaoka (Pencocokan *Pattern*)
1	`int search(unsigned char *x, int m, unsigned char *y, int n) {`
2	`   int i, j, ztBc[SIGMA][SIGMA], bmGs[XSIZE], count;`
	`   /* Preprocessing */`
3	`   BEGIN_PREPROCESSING`
4	`   preZtBc(x, m, ztBc);`
5	`   preBmGs(x, m, bmGs);`
6	`   for(i=0; i<m; i++)`
7	`      y[n+i]=y[n+m+i]=x[i];`
8	`   END_PREPROCESSING`
	`   /* Searching */`
9	`   BEGIN_SEARCHING`
10	`   count = 0;`
11	`   j = 0;`
12	`   while (j <= n - m) {`
13	`      i = m - 1;`
14	`      while (i >=0 && x[i] == y[i + j])`
15	`         --i;`
16	`      if (i < 0) {`
17	`         OUTPUT(j);`
18	`         j += bmGs[0];`
19	`      }`
20	`      else`
21	`         j += MAX(bmGs[i],ztBc[y[j + m - 2]][y[j + m - 1]]);`
22	`   }`
23	`   END_SEARCHING`
24	`   return count;`
25	`}`

1. Mengubah *source code* menjadi *Flow Graph*

Gambar 4.3 Flow Graph Algoritma Zhu Takaoka (fase pencocokan string) ini menunjukan *flow graph* algoritma *Zhu Takaoka* dalam melakukan fase pencocokkan *string* berdasarkan *pseudocode*.



Gambar 4.3 *Flow Graph* Algoritma *Zhu Takaoka* (fase pencocokkan *string*)

Keterangan :

Node = Menggambarkan Kondisi

Edge = Menggambarkan Aksi

2. Pengujian

a) Region

Region = 6

b) Cylomatic Complexity V(G)

Dari *flow graph* di atas, dapat dihitung *Cylomatic Complexity*, yakni *Region* = 6 yang dapat dari perhitungan dibawah ini.

$V_{(g)}$ = *Edge – Node* + 2

$V_{(g)}$ = 29 - 25 + 2

$V_{(g)}$ = 6

c) Independent Path

*Path*1 = 1-2-3-4-5-6-7-8-9-10-11-12-23-24-25

*Path*2 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-23-24-25

*Path*3 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-16-20-22-12-23-24-25

*Path*4 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-15-14-16-20-22-22-12-23-24-25

*Path*5 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-15-14-16-17-18-19-20-22-12-23-24-25

Path6 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-15-14-16-17-18-19-20-21-22-12-23-24-25

d) Predicate Node

$V_{(G)}$ = jumlah *node* yang memiliki lebih dari satu jalur + 1

$V_{(G)}$ = 5 + 1

$V_{(G)}$ = 6

e) Graph Matric

Penomoran ulang *flowgraph* atau penyederhanaan berdasarkan kondisi atau simpul dapat dilihat pada *Gambar 4.4 Graph Matric Algoritma Zhu Takaoka.*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	N(E)-1
1		1																								0
2			1																							0
3				1																						0
4					1																					0
5						1																				0
6							1																			0
7						1		1																		1
8									1																	0
9										1																0
10											1															0
11												1														0
12													1										1			1
13														1												0
14															1	1										1
15														1												0
16																	1			1						1
17																		1								0
18																			1							0
19																				1						0
20																					1	1				1
21																						1				0
22												1														0
23																								1		0
24																									1	0
Sum(E) + 1																										6

Gambar 4.4 *Graph Matric Algoritma Zhu Takaoka*

Tabel 4.10 Uji Coba *Independ Path*

***Independent Path* ke**	**Penggunaan**	**Contoh data masukan**	**Yang diharapkan**	**Hasil**
1	Ketika perdeicate node 1 bernilai false	Nyi Rodiah	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
2	Ketika perdeicate node 1 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
3	Ketika perdeicate node 2 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
4	Ketika perdeicate node 3 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
5	Ketika perdeicate node 4 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil
6	Ketika perdeicate node 5 bernilai true	Sunjaya	Proses looping berhenti	[√] Berhasil [ ] Tidak Berhasil

Langkah terakhir dalam melakukan pengujian menggunakan metode *basis path testing* adalah menyiapkan kasus-kasus uji untuk mengeksekusi semua alur logika yang telah dibuat pada langkah sebelumnya. Kasus uji yang dimaksud disini adalah dengan cara memberikan nilai pada variabel yang terlibat. Nilai yang dimasukan haruslah nilai yang mungkin muncul dan sesuai dengan tipe data yang telah didefinisikan. *Test case* dibuat dalam bentuk tabel yang tujuannya untuk mempermudah eksekusi setiap *basis path,* karena nilai-nilai yang dimasukan dan yang dihasilkan akan lebih mudah diamati.

1) Pengujian *Path*1 = 1-2-3-4-5-6-7-8-9-10-11-12-23-24-25

   Pada *path* ini, supaya *node* 7 dapat menuju ke *node* 8, maka nilai dari *node* 10 harus dapat menghasilkan nilai *false*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya tidak terdapat pola *Nyi Rodiah*. Agar bernilai *false* digunakan pola *Nyi Rodiah* sebagai data uji coba pencocokan pola.

2) Pengujian *Path*2 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-23-24-25

Pada *path* ini, supaya *node* 7 dapat menuju ke *node* 6, maka nilai dari *node* 7 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

3) Pengujian *Path*3 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-16-20-22-12-23-24-25

Pada *path* ini, supaya *node* 12 dapat menuju ke *node* 13, maka nilai dari *node* 12 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

4) Pengujian *Path*4 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-15-14-16-20-22-22-12-23-24-25

Pada *path* ini, supaya *node* 14 dapat menuju ke *node* 15, maka nilai dari *node* 14 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

5) Pengujian *Path*5 = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-15-14-16-17-18-19-20-22-12-23-24-25

Pada *path* ini, supaya *node* 16 dapat menuju ke *node* 17, maka nilai dari *node* 16 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

6) Pengujian *Path6* = 1-2-3-4-5-6-7-6-7-8-9-10-11-12-13-14-15-14-16-17-18-19-20-21-22-12-23-24-25

Pada *path* ini, supaya *node* 20 dapat menuju ke *node* 21, maka nilai dari *node* 20 harus dapat menghasilkan nilai *true*. Maka data uji coba diasumsikan terdapat sinopsis buku dengan jumlah 1000 karakter dan didalamnya terdapat pola *Sunjaya*. Agar bernilai *true* digunakan pola *Sunjaya* sebagai data uji coba pencocokan pola.

3. Kesimpulan

Berdasarkan pengujian pada setiap metode, dihasilkan nilai *Cyclomatic Complexity* yang sama yaitu 6. Maka dapat disimpulkan bahwa pengujian *white box* pada algoritma *Zhu Takaoka* berjalan dengan baik, karena setiap pengujian menghasilkan nilai yang sama.

4.2.2 Skenario Pengujian Akurasi

Pada pengujian akurasi kedua algoritma ini akan di uji dengan data berupa beberapa kalimat yang terdapat pada salah satu sinopsis buku berbahasa sunda yang diambil dari Rumah Baca Buku Sunda.

4.2.2.1 Pengujian Akurasi

Dilakukan uji coba terhadap beberapa kalimat yang terdapat pada salah satu data sinopsis buku yang berjudul “Pasukan Siluman Haji Prawatasari (Bagian 2)”. Pengujian ini dilakukan untuk mengetahui seberapa akurat kedua algoritma tersebut dalam menemukan pola yang dicocokan. Pengujian akan dilakukan dengan panjang *pattern* yang dimulai dari 2 sampai 20 karakter. Dapat dilihat pada Tabel 4.10 *Hasil pengujian Akurasi Algoritma Reverse Colussi dan Zhu Takaoka.*

Tabel 4.11 Hasil Pengujian Akurasi Algoritma *Reverse Colussi dan Zhu Takaoka*

Sumber *String*	***Panjang Pattern***	**Akurasi Algoritma *Zhu Takaoka***			**Akurasi Algoritma *Reverse Colussi***		
		Pola dalam Sumber String	**Pola yang ditemukan sistem**	**Akurasi**	**Pola dalam Sumber String**	**Pola yang ditemukan sistem**	**Akurasi**
samemeh milu bajuang, sanaya menta dikawinkeun heula ka nyi rodiah. ngan bae samemeh dirapalan, bet ngurunyung ki maskara. ari ki maskara teh balad pajuang ti priangan, bapana sunjaya atawa akina nyi rodiah, hartina ku kadatangan ki maskara, kedok nu dipake ku sanaya teh kabuka di hareupeun nyi rodiah pisan.	ki	4	4	100 %	4	4	100 %
	nyi	3	3	100 %	3	3	100 %
	milu	1	1	100 %	1	1	100 %
	balad	1	1	100 %	1	1	100 %
	sanaya	2	2	100 %	2	2	100 %
	maskara	3	3	100 %	3	3	100 %
	priangan	1	1	100 %	1	1	100 %
	dirapalan	1	1	100 %	1	1	100 %
	ki maskara	3	3	100 %	3	3	100 %
	dikawinkeun	1	1	100 %	1	1	100 %
	sanaya menta	1	1	100 %	1	1	100 %
	ka nyi rodiah	1	1	100 %	1	1	100 %
	bapana sunjaya	1	1	100 %	1	1	100 %
	kedok nu dipake	1	1	100 %	1	1	100 %
	akina nyi rodiah	1	1	100 %	1	1	100 %
	samemeh dirapalan	1	1	100 %	1	1	100 %
	ari ki maskara teh	1	1	100 %	1	1	100 %
	pajuang ti priangan	1	1	100 %	1	1	100 %
	samemeh milu bajuang	1	1	100 %	1	1	100 %

Untuk mengisisi kolom akurasi digunakan formula :

$$\text{Akurasi} = \frac{\text{Pola yang ditemukan sistem}}{\text{Pola dalam Sumber String}} \text{ x } 100\%$$

4.2.3 Kesimpulan Hasil Pengujian Akurasi

Berdasarkan hasil pengujian akurasi yang dilakukan oleh perangkat lunak menggunakan sumber *string* berupa potongan beberapa kalimat yang diambil dari salah satu sinopsis dengan panjang *pattern* yang diuji sepanjang 2 sampai 20 karakter ditemukan bahwa algoritma *Reverse Colussi* dan algoritma *Zhu Takaoka* memiliki tingkat akurasi atau ketepatan dalam menemukan pola atau *pattern* yang terdapat pada sumber *string* saat melakukan pencocokan *string* sebanyak 100%. Sehingga dapat disimpulkan bahwa algoritma *Reverse Colussi* dan algoritma *Zhu Takaoka* sangat cocok digunakan dalam proses pencocokan *string*.

4.2.4 Skenario Pengujian Performansi

Pada pengujian performansi kedua algoritma ini akan di uji dengan data berupa sinopsis buku berbahasa sunda yang diambil dari Rumah Baca Buku Sunda.

4.2.2.2 Pengujian Performansi

Dilakukan uji coba terhadap data sinopsis dari buku yang berjudul "Pasukan Siluman Haji Prawatasari (Bagian 2)", berformat file .txt, jumlah karakter sebanyak 454 karakter, terdapat 36 karakter tak berulang. Pengujian akan dilakukan dengan panjang *pattern* sebanyak 2 sampai 20 karakter, karakter pada pola *pattern* dipilih secara acak oleh sistem. Untuk mengetahui waktu rata-rata, waktu terbaik dan waktu terburuk yang diperlukan oleh algoritma *Reverse Colussi* dan algoritma *Zhu Takaoka*, setiap panjang *pattern* akan diuji sebanyak 500 kali pengujian. Dapat dilihat pada Tabel 4.10 *Hasil Pengujian Performansi Algoritma Reverse Colussi dan Zhu Takaoka*.

Tabel 4.12 Hasil Pengujian Performansi Algoritma *Reverse Colussi dan Zhu Takaoka*

Nama File	Jumlah Karakter pada File	Jumlah Karakter tak berulang	Panjang *Pattern*	Algoritma *Zhu Takaoka*		Algoritma *Reverse Colussi*	
				Waktu *Preprocessing*	Waktu Pencocokan	Waktu *Preprocessing*	Waktu Pencocokan
Pasukan Haji Prawatasari (Bagaian 2).txt	454	36	2	5.59 ms	0.03 ms	16.21 ms	0.13 ms
			3	5.38 ms	0.05 ms	16.12 ms	0.15 ms
			4	5.51 ms	0.04 ms	15.68 ms	0.15 ms
			5	5.62 ms	0.12 ms	17.15 ms	0.15 ms
			6	5.40 ms	0.06 ms	15.85 ms	0.13 ms
			7	5.17 ms	0.06 ms	14.29 ms	0.12 ms
			8	5.23 ms	0.02 ms	14.15 ms	0.07 ms
			9	5.87 ms	0.03 ms	14.84 ms	0.11 ms
			10	5.90 ms	0.06 ms	14.86 ms	0.10 ms
			11	6.56 ms	0.05 ms	14.83 ms	0.08 ms
			12	5.74 ms	0.05 ms	13.91 ms	0.18 ms
			13	6.51 ms	0.07 ms	15.32 ms	0.15 ms
			14	5.70 ms	0.06 ms	15.15 ms	0.14 ms
			15	5.81 ms	0.02 ms	15.24 ms	0.11 ms
			16	5.32 ms	0.07 ms	15.90 ms	0.16 ms
			17	4.65 ms	0.05 ms	14.29 ms	0.10 ms
			18	4.15 ms	0.05 ms	14.32 ms	0.12 ms
			19	4.75 ms	0.03 ms	13.59 ms	0.08 ms
			20	4.34 ms	0.07 ms	14.79 ms	0.15 ms
Rata-rata				**5.43 ms**	**0.05 ms**	**15.08 ms**	**0.13 ms**

4.2.5 Kesimpulan Hasil Pengujian Performansi

Berdasarkan hasil pengujian terhadap perangkat lunak dapat ditarik kesimpulan bahwa perangkat lunak dapat berfungsi sesuai yang diharapkan serta perangkat lunak yang dibangun telah cukup memenuhi tujuan awal pembangunan. Algoritma *Reverse Colussi* cenderung lebih lambat dalam pencocokan pola menggunakan data sinopsis buku berbahasa sunda dari Rumah Baca Buku Sunda dibandingkan dengan algoritma *Zhu-Takaoka*.

Pada pengujian salah satu data sinopsis dari buku yang berjudul "Pasukan Siluman Haji Prawatasari (Bagian 2)" dan berformat .txt, dilakukan pengujian dengan jumlah *pattern* yang berbeda, yaitu 2 sampai 20 karakter dan setiap panjang karakter diuji sebanyak 500 kali percobaan. Karakter pada *pattern* yang digunakan untuk pengujian, dipilih secara acak oleh sistem. Algoritma *Reverse Colussi* membutuhkan waktu rata-rata *preprocessing* selama 15.08 ms dan waktu rata-rata pencocokan *pattern* selama 0.13 ms sedangkan Algoritma *Zhu Takaoka* membutuhkan waktu rata-rata *preprocessing* selama 5.43 ms dan waktu rata-rata pencocokan *pattern* selama 0.05 ms.

Dari hasil pengujian tersebut terlihat bahwa waktu rata-rata yang digunakan algoritma *Reverse Colussi* pada saat *preprocessing* lebih lama dibandingkan dengan waktu rata-rata *preprocessing* yang digunakan oleh algoritma *Zhu Takaoka*, begitupun pada saat pencocokan, rata-rata waktu yang digunakan oleh algoritma *Reverse Colussi* lebih banyak dibandingkan dengan algoritma *Zhu Takaoka*, meskipun perbedaanya tidak terlalu jauh, akan tetapi pengujian tersebut sudah bisa membuktikan bahwa algoritma *Zhu Takaoka* lebih cepat dalam melakukan pencocokan pola menggunakan data sinopis buku berbahasa sunda yang diambil dari Rumah Baca Buku Sunda.