BAB IV
IMPLEMENTASI DAN PENGUJIAN SISTEM

IV.1 Implementasi

Tahap implementasi merupakan tahap penciptaan perangkat lunak dan juga tahap kelanjutan dari kegiatan perancangan aplikasi, tahap ini merupakan tahap dimana aplikasi siap untuk dioperasikan. Aplikasi ini diimplementasikan dengan nama "*Game Algojo*" dengan *file extension*-nya "*.html"*.

IV.1.1 Implementasi Perangkat Keras

Perangkat keras yang dapat digunakan untuk menjalankan aplikasi *game* ini tidak harus memiliki spesifikasi yang tinggi. Spesifikasi perangkat keras minimal yang digunakan untuk implementasi sebagai berikut :

Tabel IV.1 Implementasi Perangkat Keras

Nama Perangkat Keras	Spesifikasi
Processor	Intel Pentium 4 1,80 GHz
RAM	512 MB
VGA	Onboard Intel Grapichs
Mouse	Standar
Keyboard	Standar
Speaker	Standar

IV.1.2 Implementasi Perangkat Lunak

Perangkat lunak yang dapat digunakan untuk menjalankan aplikasi *game* ini minimal harus memiliki spesifikasi sebagai berikut :.

Tabel IV.2 Implementasi Perangkat Lunak

Nama Perangkat Lunak	Spesifikasi
Sistem Operasi	Microsoft Windows XP Profesional
Browser	Google Chrome

IV.1.3 Implementasi Antarmuka

Implementasi antarmuka ini agar dapat mengetahui performansi dari kedua algoritma. Berikut ini merupakan antarmuka yang ada pada aplikasi *game algojo* :

IV.1.3.1 Antarmuka Menu Utama

Saat pengguna membuka aplikasi pertama kali, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka menu utama. Berikut ini merupakan gambar dari antarmuka menu utama yang dapat dilihat pada gambar IV.1 :



Gambar IV.1 Antarmuka Menu Utama

Gambar IV.1 merupakan menu utama pada *game algojo*. Terdapat 3 (tiga) sub menu di dalamnya, yaitu menu *Karp-Rabin* yang berfungsi untuk menampilkan permainan menggunakan algoritma *Karp-Rabin*, menu *Reverse Colussi* yang berfungsi untuk menampilkan permainan menggunakan algoritma *Reverse Colussi* dan menu keluar yang berfungsi untuk keluar dari permainan.

IV.1.3.2 Antarmuka Menu *Karp-Rabin*

Saat pengguna memilih menu *Karp-Rabin*, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka dari implementasi *game Algojo* dengan menggunakan algoritma *Karp-Rabin*. Berikut ini adalah gambar antarmuka menu *Karp-Rabin* pada gambar IV.2 :



Gambar IV.2 Antarmuka Menu *Karp-Rabin* Memilih *New Game*



Gambar IV.3 Antarmuka Menu *Karp-Rabin* Mulai Bermain

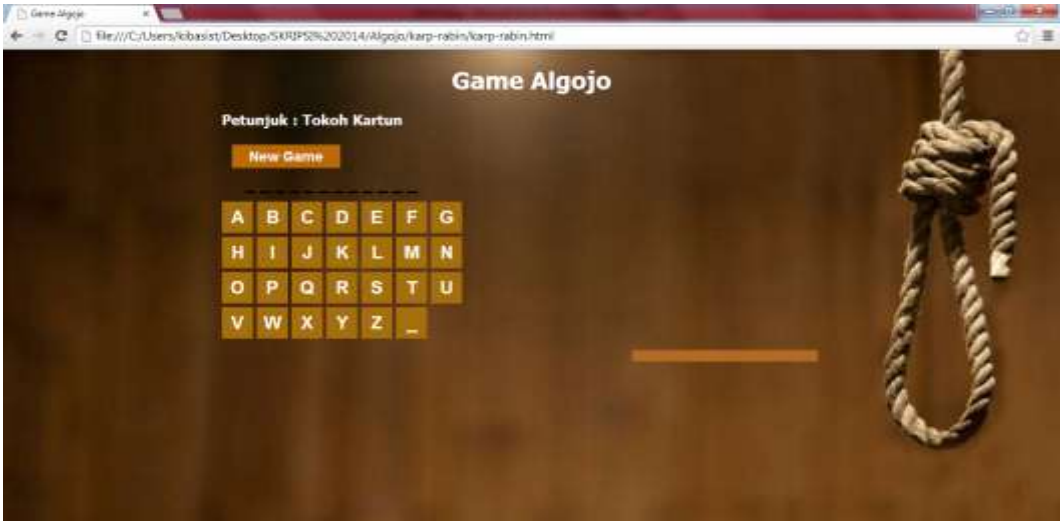
Gambar IV.3 merupakan menu Karp-Rabin pada *game Algojo*. Pada menu ini pemain akan memainkan permainan Algojo dimana algoritma pencariannya menggunakan algoritma Karp-Rabin dan untuk perbandingannya akan dibandingkan performansi dari algoritma Karp-Rabin dengan parameter kecepatan waktu proses dan pemakaian memory dalam 1 kali proses pencarian.

IV.1.3.3 Antarmuka Menu *Reverse Colussi*

Saat pengguna memilih menu *Reverse Colussi*, maka antarmuka yang ditampilkan oleh sistem adalah antarmuka dari implementasi *game Algojo* dengan menggunakan algoritma *Reverse Colussi*. Berikut ini adalah gambar antarmuka menu *Reverse Colussi* dilihat pada gambar IV.4 :



Gambar IV.4 Antarmuka Menu *Reverse Colussi* Memilih *New Game*



Gambar IV.5 Antarmuka Menu *Reverse Colussi* Mulai Bermain

Gambar IV.5 merupakan menu Reverse Colussi pada *game Algojo*. Pada menu ini pemain akan memainkan permainan Algojo dimana algoritma pencariannya menggunakan algoritma Reverse Colussi dan untuk perbandingannya akan dibandingkan performansi dari algoritma Reverse Colussi dengan parameter kecepatan waktu proses dan pemakaian memory dalam 1 kali proses pencarian.

IV.2 Pengujian Sistem

Pengujian sistem dilakukan bertujuan untuk menemukan kesalahan atau kekurangan pada penerapan algoritma yang diuji. Pengujian bermaksud untuk mengetahui implementasi dari perbandingan kedua algoritma yang dibuat sudah memenuhi kriteria yang sesuai dengan tujuan implementasi perbandingan kedua algoritma. Pengujian yang akan digunakan pada penelitian ini adalah dengan menggunakan pengujian *alpha*.

Pengujian alpha adalah pengujian yang berfokus pada persyaratan fungsional aplikasi atau perangkat lunak. Pengujian dilakukan terhadap aplikasi *game Algojo* yang sebelumnya telah dirancang dan diterapkan algoritma *Karp-Rabin* dan *Reverse Colussi*. Untuk memastikan bahwa perancangan dan penerapan kedua algoritma dapat berjalan dengan benar sesuai kebutuhan dan tujuan yang diharapkan.

IV.2.1 Pengujian Alpha

Pengujian dilakukan terhadap aplikasi untuk memastikan bahwa aplikasi dapat berjalan dengan benar sesuai dengan kebutuhan dan tujuan yang diharapkan. Proses pengujian terhadap aplikasi akan menggunakan *black box testing dan white box testing*. Pengujian *black box* berfokus pada persyaratan *fungsional* perangkat lunak. Berikut ini skenario pengujian dari *game Algojo* dapat dilihat pada tabel IV.1 :

Tabel IV.3 Skenario Pengujian *Alpha*

No	Komponen yang diuji	Skenario Pengujian	Jenis Pengujian
1.	Menu Utama	Memilih menu *Karp-Rabin*	*Black Box*
		Memilih menu *Reverse Colussi*	*Black Box*
		Memilih menu keluar	*Black Box*
2.	*YOU WIN*	Memilih tombol *New Game*	*Black Box*
		Memilih tombol keluar	*Black Box*
3.	*YOU LOSE*	Memilih tombol *New Game*	*Black Box*
		Memilih tombol keluar	*Black Box*
4.	Uji Algoritma	Proses Pencocokan algoritma Karp-Rabin	*White Box*
		Proses Pencocokan Algoritma Reverse Colussi	*White Box*

IV.2.1.1 Kasus dan Hasil Pengujian (*Black Box Testing*)

Berikut ini adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *black box* berdasarkan pengamatan yang sudah dilakukan dan menghasilkan kesimpulan dari pengujian tersebut.

1. Pengujian Menu Utama

Pengujian menu utama dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel IV.4 :

Tabel IV.4 Pengujian Menu Utama

Kasus yang akan di uji	Skenario dan Hasil Uji		
	Skenario Uji	Hasil yang diharapkan	Kesimpulan
Menu Utama	Memilih menu *Karp-Rabin*	Sistem menampilkan halaman menu *Karp-Rabin*	[✓] Berhasil [ ] Tidak Berhasil
	Memilih menu Reverse Colussi	Sistem menampilkan halaman menu Reverse Colussi	[✓] Berhasil [ ] Tidak Berhasil
	Memilih menu keluar	Sistem menampilkan pertanyaan ingin keluar dari aplikasi atau tidak	[✓] Berhasil [ ] Tidak Berhasil

2. Pengujian *You Win*

Pengujian *You Win* dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel IV.5 :

Tabel IV.5 Pengujian *You Win*

Kasus yang akan diuji	Skenario dan Hasil Uji		
	Skenario Uji	Hasil yang diharapkan	Kesimpulan
You Win	Memilih tombol *New Game*	Sistem memainkan permainan kembali	[✓] Berhasil [ ] Tidak Berhasil
	Memilih tombol keluar	Sistem keluar dari permainan	[✓] Berhasil [ ] Tidak Berhasil

3. Pengujian *You Lose*

Pengujian *You Lose* dengan kasus dan hasil uji, kemudian diamati dan menghasilkan kesimpulan dapat dilihat pada tabel IV.6 :

Tabel IV.6 Pengujian *You Lose*

Kasus yang akan diuji	Skenario dan Hasil Uji		
	Skenario Uji	Hasil yang diharapkan	Kesimpulan
You Lose	Memilih tombol *New Game*	Sistem memainkan permainan kembali	[✓] Berhasil [ ] Tidak Berhasil
	Memilih tombol keluar	Sistem keluar dari permainan	[✓] Berhasil [ ] Tidak Berhasil

IV.2.1.2 Kasus dan Hasil Pengujian (*White Box Testing*)

Pengujian white box bertujuan untuk mengetahui kinerja logika yang dibuat pada sebuah perangkat lunak apakah berjalan dengan baik atau tidak. Pengujian *white box* akan digunakan pada algoritma *Reverse Colussi* dan *Karp Rabin*, untuk mengukur kinerja logika berdasarkan *pseudocode* yang telah dibuat pada tahap analisis.

Pengujian *white box* dilakukan dengan beberapa tahap, yaitu :

1. Ubah *pseudocode* 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*.

IV.2.1.2.1 *White Box* Algoritma *Karp-rabin*

Fase Menentukan fungsi hash

Berikut ini adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *White Box* dari algoritma *Karp-Rabin*

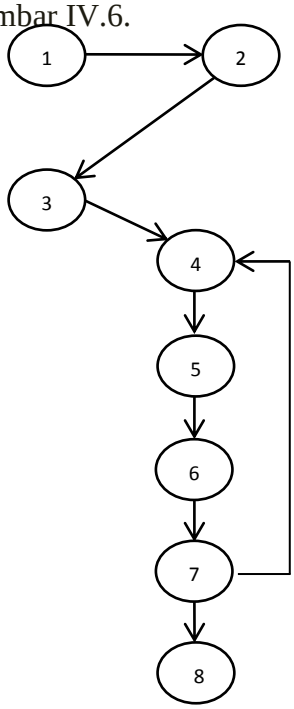
berdasarkan *Source Code* algoritma pencariannya dengan menentukan fungsi hash.

Deklarasi	
1	`function hashFromTo(str, from, to)`
2	`var hash = 0;`
Fungsi Penentuan nilai *hash*	
3	`{`
4	`  for (var i = from; i < to && i < str.length; i++) {`
5	`    hash = primeBase * hash + str.charCodeAt(i);`
6	`    }`
7	`    return hash;`
8	`}`

Gambar IV.6 *White Box Karp-Rabin* fase menentukan fungsi *hash*

1. *Flow Graph*

Berikut ini merupakan *flow graph Karp-rabin* menentukan fungsi *hash* berdasarkan *pseudocode* pada Gambar IV.6.



Gambar IV.7 *Flow Graph* Algoritma *Karp-Rabin* menentukan fungsi *hash*

Keterangan :

 = Menggambarkan Kondisi

 = Menggambarkan Aksi

2. *Region*

Region = 2

3. *Cyclomatic Complexity V(G)*

Dari *flow graph* di atas, dapat dihitung *cyclomatic complexity*, yakni *Region* = 2 yang di dapat dari perhitungan dibawah ini:

E = 8 N = 8

V(G) = E – N + 2

V(G) = 8 *edge* – 8 *node* + 2

V(G) = 0 + 2 = 2

Dimana:

E = jumlah *edge*

N = jumlah *node*

4. *Independent Path*

Dari hasil perhitungan *cyclomatic complexity* terdapat 5 *independent path* yaitu:

Path 1 = 1-2-3-4-5-6-7-4

Path 2 = 1-2-3-4-5-6-7-4-5-6-7-8

5. *Graph Matric*

Penomoran ulang *flowgraph* atau penyederhanaan berdasarkan kondisi atau simpul dapat dilihat pada Gambar IV.8

	1	2	3	4	5	6	7	8	N(E)-1
1		1							1-1 = 0
2			1						1-1 = 0
3				1					1-1 = 0
4					1				1-1 = 0
5						1			1-1 = 0
6							1		1-1 = 0
7				1				1	2-1 = 1
Sum(E) +1									1+1 = 2

Gambar IV.8 *Graph Matric* Algoritma *Karp-Rabin* menentukan fungsi *hash*

Keterangan :

1. Baris dan kolom merepresentasikan simpul.
2. Nilai 1 merepresentasikan adanya keterhubungan antar node/simpul. yang dibuat sesuai alur logika dari program tersebut. penggunaan kondisi mempengaruhi jumlah node dan edge dari *flow graph* tersebut.
   - Baris 1 dan kolom 2 = merepresentasikan adanya keterhubungan antar node/simpul 1 dan node/simpul 2 dari *flow graph*.
   - Baris 2 dan kolom 3 = merepresentasikan adanya keterhubungan antar node/simpul 2 dan node/simpul 3 dari *flow graph*.
   - Baris 3 dan kolom 4 = merepresentasikan adanya keterhubungan antar node/simpul 3 dan node/simpul 4 dari *flow graph*.
   - Baris 4 dan kolom 5 = merepresentasikan adanya keterhubungan antar node/simpul 4 dan node/simpul 5 dari *flow graph*.
   - Baris 5 dan kolom 6 = merepresentasikan adanya keterhubungan antar node/simpul 5 dan node/simpul 6 dari *flow graph*.
   - Baris 6 dan kolom 7 = merepresentasikan adanya keterhubungan antar node/simpul 6 dan node/simpul 7 dari *flow graph*.

- Baris 7 dan kolom 8 , kolom 4 = merepresentasikan adanya keterhubungan antar node/simpul 7 , node/simpul 8 dan node/simpul 4 dari *flow graph*.

Fase pencocokan nilai *hash*

Berikut ini adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *White Box* dari algoritma *Karp-Rabin* berdasarkan *Source Code* algoritma pencariannya dengan melakukan pencocokan nilai hash yang di dapat.

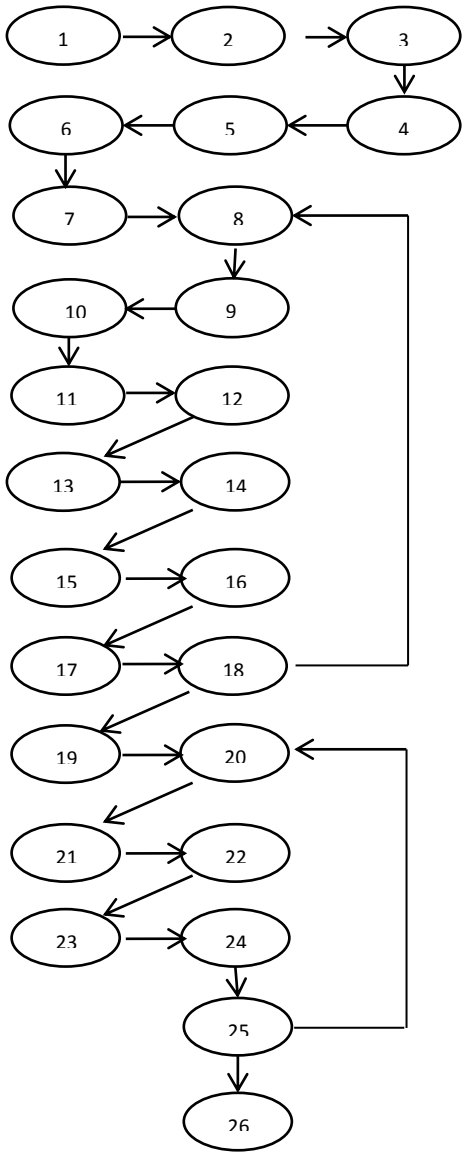
Deklarasi	
1	Function searchRabinKarp()
2	Var matches = [];
3	Var matches = true;
Fungsi pencarian Karp-Rabin	
4	var hashStr = hashFromTo(str, 0, str.length);
5	var hashTextPart = hashFromTo(text, 0, str.length);
6	var primeToPower = Math.pow(primeBase, str.length);
7	var maxIndexForPotentialMatch = text.length - str.length;
8	for (var i = 0; i <= maxIndexForPotentialMatch; i++) {
9	console.log("");
10	if (hashTextPart === hashStr) {
11	if (matchesAtIndex(i, text, str)) {
12	matches.push(i);
13	}
14	}
15	hashTextPart = primeBase * hashTextPart - primeToPower * text.charCodeAt(i) + text.charCodeAt(i + str.length);
16	}
17	return matches;
18	{
19	for (var j = 0; j < str.length; j++) {
20	if (text[index + j] !== str[j]) {
21	matches = false;
22	break;

```
23      }
24    }
25    return matches;
26  }
```

Gambar IV.9 *White Box Karp-Rabin* fase pencocokan nilai hash

1. *Flow Graph*

Berikut ini merupakan *flow graph Karp-rabin* pencocokan nilai *hash* berdasarkan *pseudocode* pada Gambar IV.10.



Gambar IV.10 *Flow Graph* Algoritma *Karp-Rabin* pencocokan nilai *hash*

Keterangan :

 = Menggambarkan Kondisi

 = Menggambarkan Aksi

6. *Region*

Region = 3

7. *Cyclomatic Complexity V(G)*

Dari *flow graph* di atas, dapat dihitung *cyclomatic complexity*, yakni *Region* = 5 yang di dapat dari perhitungan dibawah ini:

E = 27 N = 26

V(G) = E – N + 2

V(G) = 27 *edge* – 26 *node* + 2

V(G) = 1 + 2 = 3

Dimana:

E = jumlah *edge*

N = jumlah *node*

8. *Independent Path*

Dari hasil perhitungan *cyclomatic complexity* terdapat 5 *independent path* yaitu:

Path 1 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-8

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

Path 3 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-18-8-9-10-11-12-14-15-16-17-18-19-20-21-22-23-24-25-20-21-22-23-24-25-26

9. *Graph Matric*

Penomoran ulang *flowgraph* atau penyederhanaan berdasarkan kondisi atau simpul dapat dilihat pada Gambar IV.11

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

Gambar IV.11 *Graph Matric* Algoritma *Karp-Rabin* pencocokan nilai *hash*

Keterangan :

1. Baris dan kolom merepresentasikan simpul.
2. Nilai 1 merepresentasikan adanya keterhubungan antar node/simpul. yang dibuat sesuai alur logika dari program tersebut. penggunaan kondisi memengaruhi jumlah node dan edge dari *flow graph* tersebut.
   - Baris 1 dan kolom 2 = merepresentasikan adanya keterhubungan antar node/simpul 1 dan node/simpul 2 dari *flow graph*.
   - Baris 2 dan kolom 3 = merepresentasikan adanya keterhubungan antar node/simpul 2 dan node/simpul 3 dari *flow graph*.
   - Baris 3 dan kolom 4 = merepresentasikan adanya keterhubungan antar node/simpul 3 dan node/simpul 4 dari *flow graph*.
   - Baris 4 dan kolom 5 = merepresentasikan adanya keterhubungan antar node/simpul 4 dan node/simpul 5 dari *flow graph*.
   - Baris 5 dan kolom 6 = merepresentasikan adanya keterhubungan antar node/simpul 5 dan node/simpul 6 dari *flow graph*.
   - Baris 6 dan kolom 7 = merepresentasikan adanya keterhubungan antar node/simpul 6 dan node/simpul 7 dari *flow graph*.
   - Baris 7 dan kolom 8 = merepresentasikan adanya keterhubungan antar node/simpul 7 dan node/simpul 8 dari *flow graph*.
   - Baris 8 dan kolom 9 = merepresentasikan adanya keterhubungan antar node/simpul 8 dan node/simpul 9 dari *flow graph*.
   - Baris 9 dan kolom 10 = merepresentasikan adanya keterhubungan antar node/simpul 9 dan node/simpul 10 dari *flow graph*.
   - Baris 10 dan kolom 11 = merepresentasikan adanya keterhubungan antar node/simpul 10 dan node/simpul 11 dari *flow graph*.
   - Baris 11 dan kolom 12 = merepresentasikan adanya keterhubungan antar node/simpul 11 dan node/simpul 12 dari *flow graph*.
   - Baris 12 dan kolom 13 = merepresentasikan adanya keterhubungan antar node/simpul 12 dan node/simpul 13 dari *flow graph*.
   - Baris 13 dan kolom 14 = merepresentasikan adanya keterhubungan antar node/simpul 13 dan node/simpul 14 dari *flow graph*.

- Baris 14 dan kolom 15 = merepresentasikan adanya keterhubungan antar node/simpul 14 dan node/simpul 15 dari *flow graph*.
- Baris 15 dan kolom 16 = merepresentasikan adanya keterhubungan antar node/simpul 15 dan node/simpul 16 dari *flow graph*.
- Baris 16 dan kolom 17 = merepresentasikan adanya keterhubungan antar node/simpul 16 dan node/simpul 17 dari *flow graph*.
- Baris 17 dan kolom 18 = merepresentasikan adanya keterhubungan antar node/simpul 17 dan node/simpul 18 dari *flow graph*.
- Baris 18 dan kolom 19, kolom 8 = merepresentasikan adanya keterhubungan antar node/simpul 18 , node/simpul 19 dan node/simpul 8 dari *flow graph*.
- Baris 19 dan kolom 20 = merepresentasikan adanya keterhubungan antar node/simpul 19 dan node/simpul 20 dari *flow graph*.
- Baris 20 dan kolom 21 = merepresentasikan adanya keterhubungan antar node/simpul 20 dan node/simpul 21 dari *flow graph*.
- Baris 21 dan kolom 22 = merepresentasikan adanya keterhubungan antar node/simpul 21 dan node/simpul 22 dari *flow graph*.
- Baris 22 dan kolom 23 = merepresentasikan adanya keterhubungan antar node/simpul 22 dan node/simpul 23 dari *flow graph*.
- Baris 23 dan kolom 24 = merepresentasikan adanya keterhubungan antar node/simpul 23 dan node/simpul 24 dari *flow graph*.
- Baris 24 dan kolom 25 = merepresentasikan adanya keterhubungan antar node/simpul 24 dan node/simpul 25 dari *flow graph*.
- Baris 25 dan kolom 26, kolom 20 = merepresentasikan adanya keterhubungan antar node/simpul 25, node/simpul 26 dan node/simpul 20 dari *flow graph*.

IV.2.1.2.2 *White Box* Algoritma *Reverse Colussi*

Fase Pemrosesan Awal

Berikut ini adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *White Box* dari algoritma *Reverse Colussi* berdasarkan *Source Code* algoritmanya dan menghasilkan kesimpulan dari pengujian tersebut.

	Deklarasi
1	function rcBcrcGs ()
2	var rcBc[ASIZE][XSIZE];
3	var rcGs[];
4	var a;
5	var i;
6	var j;
7	var k;
8	var q;
9	var r;
10	var s;
11	var hmin[XSIZE];
12	var kmin[XSIZE];
13	var link[XSIZE];
14	var locc[ASIZE];
15	var rmin[XSIZE];
	Fase pemrosesan Awal
16	{
	//Menentukan link dan locc
17	for (a = 0; a < ASIZE; ++a)
18	locc[a] = -1;
19	link[0] = -1;
20	for (i = 0; i < m - 1; ++i) {
21	link[i + 1] = locc[x[i]];
22	locc[x[i]] = i;
23	}

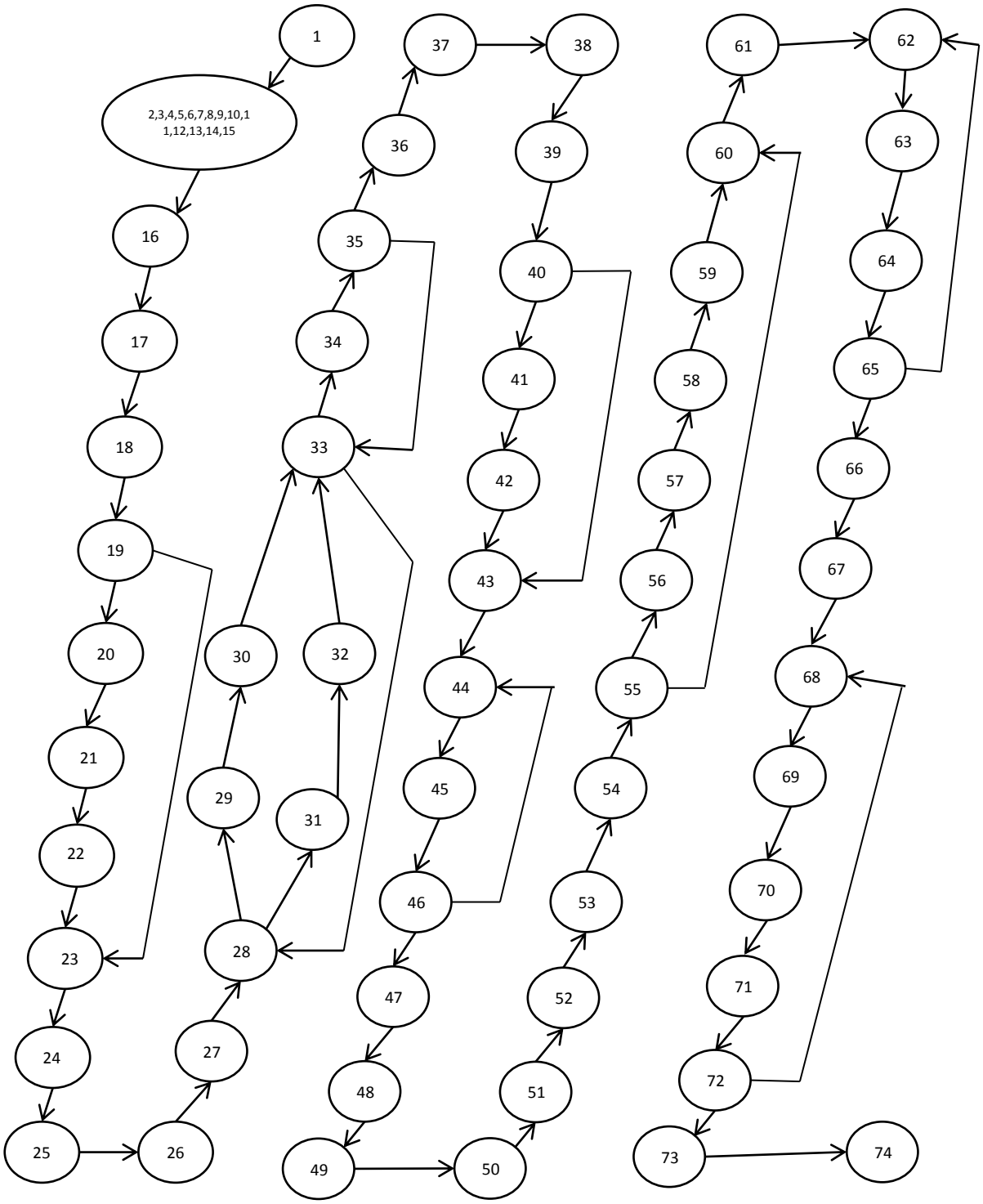
```
    //Menentukan tabel rcBc
24  for (a = 0; a < ASIZE; ++a)
25     for (s = 1; s <= m; ++s) {
26        i = locc[a];
27        j = link[m - s];
28        while (i - j != s && j >= 0)
29           if (i - j > s)
30              i = link[i + 1];
31           else
32              j = link[j + 1];
33        while (i - j > s)
34           i = link[i + 1];
35        rcBc[a][s] = m - i - 1;
36     }
    //Menentukan hmin
37  k = 1;
38  i = m - 1;
39  while (k <= m) {
40     while (i - k >= 0 && x[i - k] == x[i])
41        --i;
42     hmin[k] = i;
43     q = k + 1;
44     while (hmin[q - k] - (q - k) > i) {
45        hmin[q] = hmin[q - k];
46        ++q;
47     }
48     i += (q - k);
49     k = q;
50     if (i == m)
51        i = m - 1;
52  }
    //Menentukan kmin
53  memset(kmin, 0, m * sizeof(int));
54  for (k = m; k > 0; --k)
55     kmin[hmin[k]] = k;
    //Menentukan rmin
```

```
56      for (i = m - 1; i >= 0; --i) {
57         if (hmin[i + 1] == i)
58            r = i + 1;
59         rmin[i] = r;
60      }
        //Menentukan tabel rcGs
61      i = 1;
62      for (k = 1; k <= m; ++k)
63         if (hmin[k] != m - 1 && kmin[hmin[k]] == k) {
64            h[i] = hmin[k];
65            rcGs[i++] = k;
66         }
67      i = m-1;
68      for (j = m - 2; j >= 0; --j)
69         if (kmin[j] == 0) {
70            h[i] = j;
71            rcGs[i--] = rmin[j];
72         }
73      rcGs[m] = rmin[0];
74   }
```

Gambar IV.12 *White Box* Algoritma *Reverse Colussi* Fase Pemrosesan Awal

1. *Flow Graph*

Berikut ini merupakan *flow graph Reverse Colussi* menentukan fase pemrosesan awal berdasarkan *pseudocode* pada Gambar IV.13.



Gambar IV.13 *Flow Graph* Algoritma *Reverse Colussi* fase pemrosesan awal

Keterangan :

 = Menggambarkan Kondisi

 = Menggambarkan Aksi

2. *Region*

Region = 9

3. *Cyclomatic Complexity V(G)*

Dari *flow graph* di atas, dapat dihitung *cyclomatic complexity*, yakni *Region* = 9 yang di dapat dari perhitungan dibawah ini:

E = 68 N = 61

V(G) = E – N + 2

V(G) = 68 *edge* – 61 *node* + 2

V(G) = 7 + 2 = 9

Dimana:

E = jumlah *edge*

N = jumlah *node*

4. *Independent Path*

Dari hasil perhitungan *cyclomatic complexity* terdapat 5 *independent path* yaitu:

Path 1 = 1-2-3-4-5-6-7-8-9-10-6

Path 2 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-17-18-19-20-18

Path 3 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-16-17-20

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

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

Path 6 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-20-21-22-23-24-25-26-27-28-28-30-27

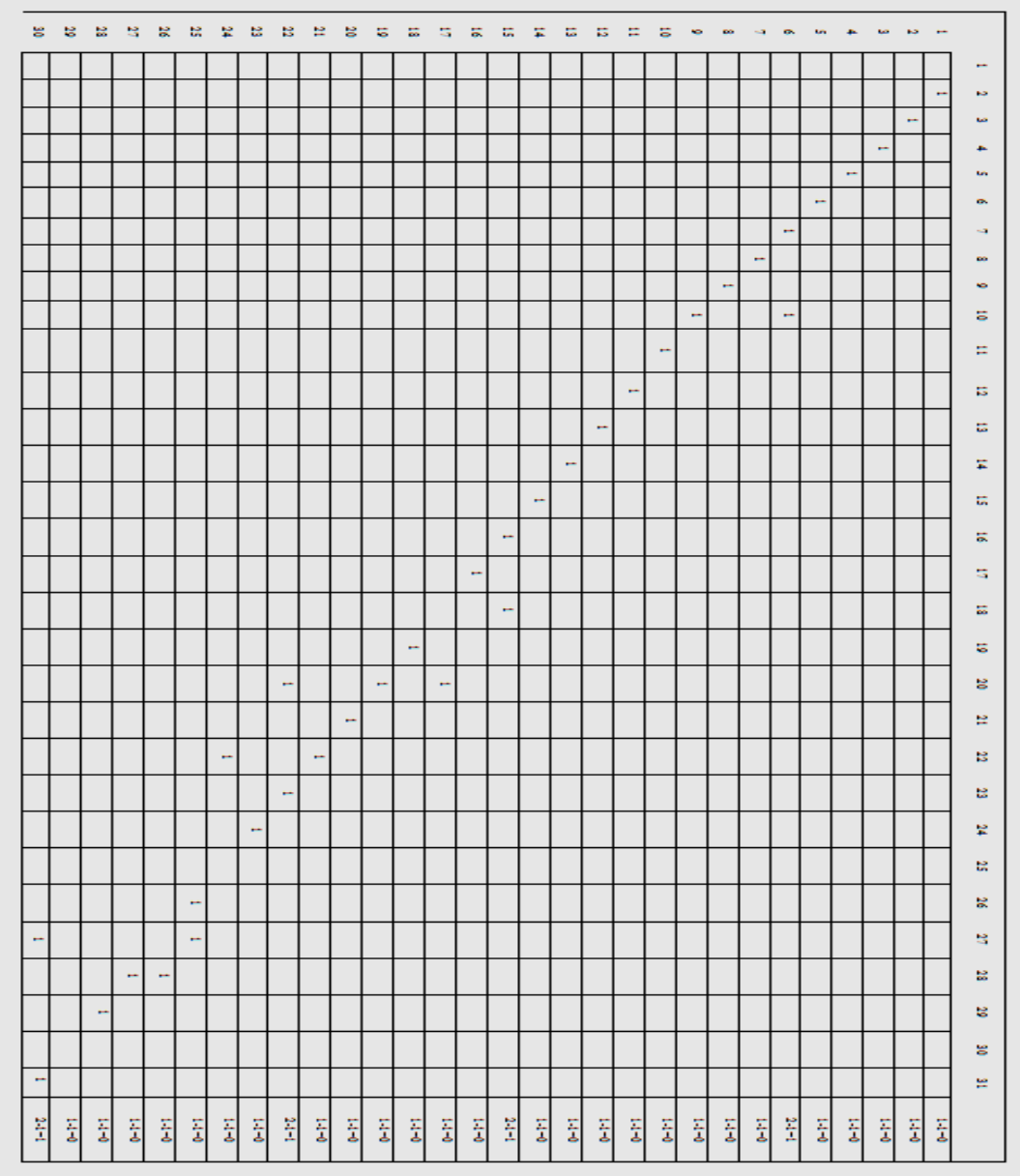
Path 7 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-20-21-22-23-24-25-26-27-28-29-30-31-32-33-31

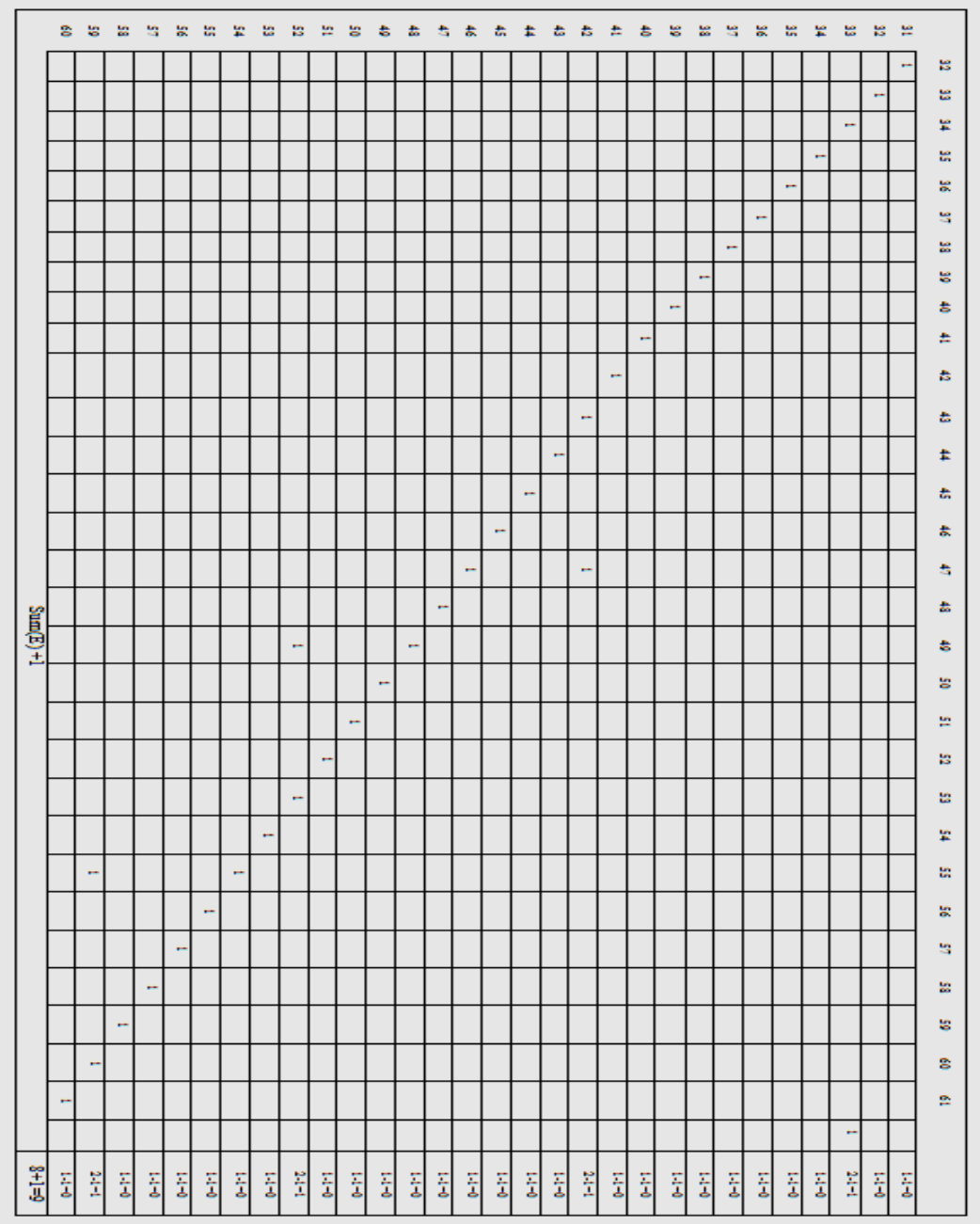
Path 8 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-42

Path 9 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-20-21-22-23-24-25-26-27-28-29-30-31-32-33-34-35-36-37-38-39-40-41-42-43-44-45-46-47-48-49-50-51-52-53-54-55-56-57-58-59-60-61

5. *Graph Matric*

Penomoran ulang *flowgraph* atau penyederhanaan berdasarkan kondisi atau simpul dapat dilihat pada gambar IV.14





Gambar IV.14. *Graph Matric* Algoritma *Reverse Colussi*

Keterangan :

1. Baris dan kolom merepresentasikan simpul.
2. Nilai 1 merepresentasikan adanya keterhubungan antar node/simpul. yang dibuat sesuai alur logika dari program tersebut. penggunaan kondisi memengaruhi jumlah node dan edge dari *flow graph* tersebut.

Fase Pencocokan Kata

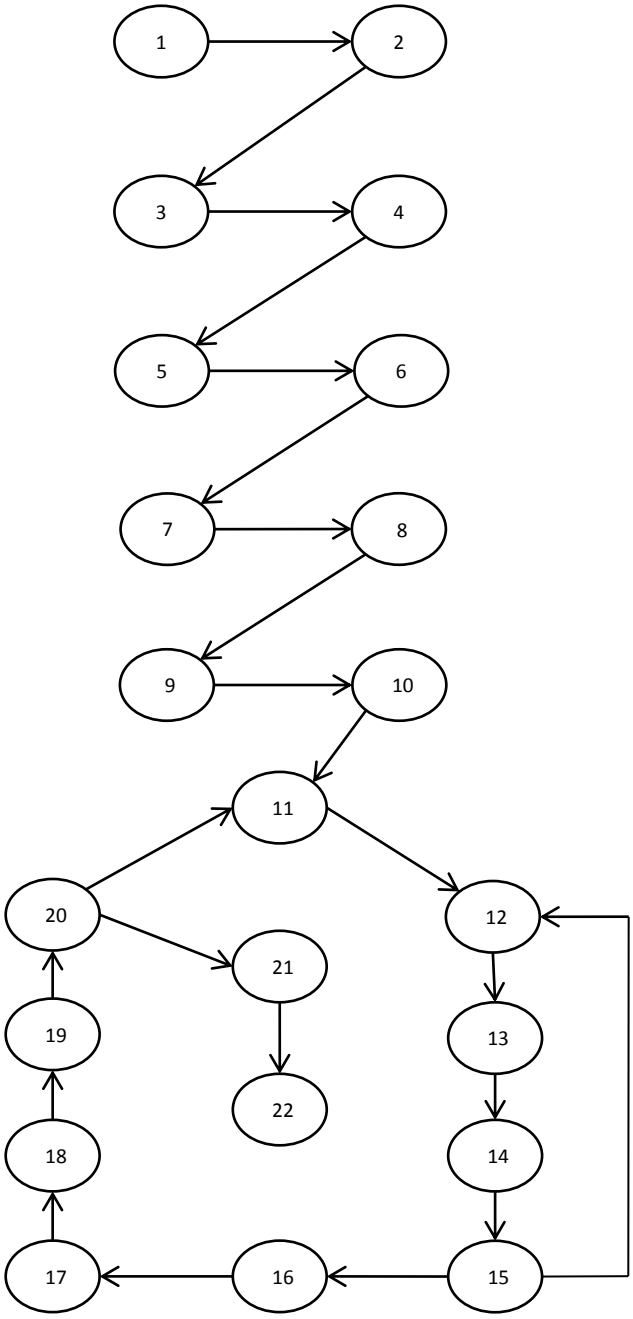
Berikut ini adalah hasil dari pengujian perangkat lunak yang sudah dibangun menggunakan metode *White Box* dari algoritma *Reverse Colussi* pada fase pencocokan kata berdasarkan *Source Code* algoritmanya.

	Deklarasi
1	Function searchReverseColussi()
2	Var i;
3	Var j;
4	Var s;
5	Var rcBc[ASIZE][XSIZE];
6	Var rcGs[XSIZE];
7	Var h[XSIZE];
	Fase Pencocokan Kata
	//Mulai Pencarian
8	{
9	j = 0;
10	s = m;
11	while (j <= n - m) {
12	while (j <= n - m && x[m - 1] != y[j + m - 1]) {
13	s = rcBc[y[j + m - 1]][s];
14	j += s;
15	}
16	for (i = 1; i < m && x[h[i]] == y[j + h[i]]; ++i);
17	if (i >= m)
18	OUTPUT(j);
19	s = rcGs[i];
20	j += s;
21	}
22	}

Gambar IV.15 *White Box* Algoritma *Reverse Colussi* Fase Pencocokan Kata

1. *Flow Graph*

Berikut ini merupakan *flow graph Reverse Colussi* menentukan fase pencocokan kata berdasarkan *pseudocode* pada Gambar IV.5.



Gambar IV.16 *Flow Graph* Algoritma *Reverse Colussi* fase pencocokan kata

Keterangan :

 = Menggambarkan Kondisi

 = Menggambarkan Aksi

2. *Region*

Region = 3

3. *Cyclomatic Complexity V(G)*

Dari *flow graph* di atas, dapat dihitung *cyclomatic complexity*, yakni *Region* = 3 yang di dapat dari perhitungan dibawah ini:

E = 23 N = 22

V(G) = E – N + 2

V(G) = 23 *edge* – 22 *node* + 2

V(G) = 1 + 2 = 3

Dimana:

E = jumlah *edge*

N = jumlah *node*

4. *Independent Path*

Dari hasil perhitungan *cyclomatic complexity* terdapat 3 *independent path* yaitu:

Path 1 = 1-2-3-4-5-6-7-8-9-10-11-12-13-14-15-12

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

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

5. *Graph Matric*

Penomoran ulang *flowgraph* atau penyederhanaan berdasarkan kondisi atau simpul dapat dilihat pada tabel IV.17

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

Gambar IV.17. *Graph Matric* Algoritma *Reverse Colussi* pencocokan kata

Keterangan :

1. Baris dan kolom merepresentasikan simpul.
2. Nilai 1 merepresentasikan adanya keterhubungan antar node/simpul. yang dibuat sesuai alur logika dari program tersebut. penggunaan kondisi memengaruhi jumlah node dan edge dari *flow graph* tersebut.
   - Baris 1 dan kolom 2 = merepresentasikan adanya keterhubungan antar node/simpul 1 dan node/simpul 2 dari *flow graph*.
   - Baris 2 dan kolom 3 = merepresentasikan adanya keterhubungan antar node/simpul 2 dan node/simpul 3 dari *flow graph*.
   - Baris 3 dan kolom 4 = merepresentasikan adanya keterhubungan antar node/simpul 3 dan node/simpul 4 dari *flow graph*.
   - Baris 4 dan kolom 5 = merepresentasikan adanya keterhubungan antar node/simpul 4 dan node/simpul 5 dari *flow graph*.

- Baris 5 dan kolom 6 = merepresentasikan adanya keterhubungan antar node/simpul 5 dan node/simpul 6 dari *flow graph*.
- Baris 6 dan kolom 7 = merepresentasikan adanya keterhubungan antar node/simpul 6 dan node/simpul 7 dari *flow graph*.
- Baris 7 dan kolom 8 = merepresentasikan adanya keterhubungan antar node/simpul 7 dan node/simpul 8 dari *flow graph*.
- Baris 8 dan kolom 9 = merepresentasikan adanya keterhubungan antar node/simpul 8 dan node/simpul 9 dari *flow graph*.
- Baris 9 dan kolom 10 = merepresentasikan adanya keterhubungan antar node/simpul 9 dan node/simpul 10 dari *flow graph*.
- Baris 10 dan kolom 11 = merepresentasikan adanya keterhubungan antar node/simpul 10 dan node/simpul 11 dari *flow graph*.
- Baris 11 dan kolom 12 = merepresentasikan adanya keterhubungan antar node/simpul 11 dan node/simpul 12 dari *flow graph*.
- Baris 12 dan kolom 13 = merepresentasikan adanya keterhubungan antar node/simpul 12 dan node/simpul 13 dari *flow graph*.
- Baris 13 dan kolom 14 = merepresentasikan adanya keterhubungan antar node/simpul 13 dan node/simpul 14 dari *flow graph*.
- Baris 14 dan kolom 15 = merepresentasikan adanya keterhubungan antar node/simpul 14 dan node/simpul 15 dari *flow graph*.
- Baris 15 dan kolom 16, kolom 11 = merepresentasikan adanya keterhubungan antar node/simpul 15 dan node/simpul 16, node/simpul 11 dari *flow graph*.
- Baris 16 dan kolom 17 = merepresentasikan adanya keterhubungan antar node/simpul 16 dan node/simpul 17 dari *flow graph*.
- Baris 17 dan kolom 18 = merepresentasikan adanya keterhubungan antar node/simpul 17 dan node/simpul 18 dari *flow graph*.
- Baris 18 dan kolom 19 = merepresentasikan adanya keterhubungan antar node/simpul 18 dan node/simpul 19 dari *flow graph*.
- Baris 19 dan kolom 20 = merepresentasikan adanya keterhubungan antar node/simpul 19 dan node/simpul 20 dari *flow graph*.

- Baris 20 dan kolom 21, kolom 11 = merepresentasikan adanya keterhubungan antar node/simpul 20 dan node/simpul 21, node/simpul 11 dari *flow graph*.
- Baris 21 dan kolom 22 = merepresentasikan adanya keterhubungan antar node/simpul 21 dan node/simpul 22 dari *flow graph*.

IV.2.2 Skenario Pengujian Performansi

Pada pengujian performansi kedua algoritma ini mendapatkan inputan yang sama yaitu DORAEMON dan DRAGON BALL dimana akan diinputkan huruf yang benar (ada di kata tersembunyi) dan huruf yang salah (tidak ada di kata tersembunyi dan akan aktifkan hukuman)

Perngujian Performansi Algoritma Karp-Rabin

Tabel IV.7 Pengujian performansi Karp-Rabin

No	Kata Yang Dicari	Looping	Uji Ke -	input	Keterangan pencocokan	Algoritma Karp-Rabin		
						Batas Looping Big-O (m+n)	Kecepatan Waktu (detik)	Memori (byte)
1	DORAEMON	8 kali	40	D	Berhasil	28	0.1499	40
			40	E	Berhasil	28	0.0200	40
			32	W	Hukuman	28	0.0300	32
			40	A	Berhasil	28	0.0300	40
			40	N	Berhasil	28	0.0300	40
			48	O	Berhasil	28	0.0400	48
			40	M	Berhasil	28	0.0499	40
			32	T	Hukuman	28	0.0500	32
			32	S	Hukuman	28	0.0399	32
			40	R	Berhasil	28	0.0800	40
RATA - RATA							**0.05197**	**38.4**
2	DRAGON BALL	11 kali	40	B	Berhasil	28	0.1699	40
			40	R	Berhasil	28	0.0500	40
			32	U	Hukuman	28	0.0400	32
			40	G	Berhasil	28	0.0400	40
			48	L	Berhasil	28	0.0300	48
			40	D	Berhasil	28	0.0400	40

			40	N	Berhasil	28	0.0500	40
			40	O	Berhasil	28	0.0600	40
			32	I	Hukuman	28	0.0500	32
			40	(spasi)	Berhasil	28	0.0400	40
			48	A	Berhasil	28	0.0699	48
RATA - RATA							**0.058164**	**40**

Perngujian Performansi Algoritma Reverse Colussi

Tabel IV.8 Pengujian performansi Reverse Colussi

No	**Kata Yang Dicari**	**Looping**	**Uji Ke -**	**Input**	**Keterangan pencocokan**	**Algoritma Reverse Colussi**		
						Batas Looping Big-O ($(m^2)+n$)	**Kecepatan Waktu (detik)**	**Memori (kb)**
1	DORAEMON	8 kali	1	D	Berhasil	55	0.1400	40
			2	E	Berhasil	55	0.0300	32
			3	W	Hukuman	55	0.0300	34
			4	A	Berhasil	55	0.0299	34
			5	N	Berhasil	55	0.0300	26
			6	O	Berhasil	55	0.0300	36
			7	M	Berhasil	55	0.0299	30
			8	T	Hukuman	55	0.0299	34
			9	S	Hukuman	55	0.0200	34
			10	R	Berhasil	55	0.0700	36
RATA - RATA							**0.04397**	**33.6**
2	DRAGON BALL	11 kali	1	B	Berhasil	55	0.1499	32
			2	R	Berhasil	55	0.0300	44
			3	U	Hukuman	55	0.0100	40
			4	G	Berhasil	55	0.0100	40
			5	L	Berhasil	55	0.0299	34
			6	D	Berhasil	55	0.0200	46
			7	N	Berhasil	55	0.0300	36
			8	O	Berhasil	55	0.0200	38
			9	I	Hukuman	55	0.0200	40
			10	(spasi)	Berhasil	55	0.0299	34
			11	A	Berhasil	55	0.0500	38
RATA - RATA							**0.36336**	**38.36**

IV.2.3 Kesimpulan Hasil Pengujian Performansi

Berdasarkan hasil pengujian terhadap perangkat lunak dapat ditarik kesimpulan perangkat lunak dapat berfungsi sesuai dengan yang diharapkan serta perangkat lunak yang dibangun telah cukup memenuhi tujuan awal pembangunan dan algoritma *Karp-Rabin* cendrung lebih lambat dalam pencarian kata dengan *single pattern* dibandingkan dengan algoritma *Reverse Colussi*. Pada pengujian kata DORAEMON, algoritma *Karp-Rabin* memerlukan rata-rata kecepatan waktu 0.05197 detik dan pemakaian memori 38.4 byte sedangkan algoritma *Reverse Colussi* memerlukan kecepatan waktu 0.04397 detik dan pemakaian memori 33.6 byte. Pada pengujian kata DRAGON BALL, algoritma *Karp-Rabin* memerlukan rata-rata kecepatan waktu 0.058164 detik dan pemakaian memori 40 byte sedangkan algoritma *Reverse Colussi* memerlukan kecepatan waktu 0.036336 detik dan pemakaian memori 38.36 byte. Meskipun perbedaan kecepatan waktunya sangat kecil tetapi tetap dinyatakan bahwa algoritma *Reverse Colussi* lebih cepat dibandingkan dengan algoritma *Karp-Rabin* dalam mencari kata dengan *single pattern.*