

**INSTRUMENTASI KODE PROGRAM SECARA OTOMATIS
UNTUK *PATH TESTING***

RADEN ASRI RAMADHINA FITRIANI



**ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
BOGOR
2018**

PERNYATAAN MENGENAI SKRIPSI DAN SUMBER INFORMASI SERTA PELIMPAHAN HAK CIPTA

Dengan ini saya menyatakan bahwa skripsi berjudul Instrumentasi Kode Program Secara Otomatis Untuk *Path Testing* adalah benar karya saya dengan arahan dari komisi pembimbing dan belum diajukan dalam bentuk apa pun kepada perguruan tinggi mana pun. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan maupun tidak diterbitkan dari penulis lain telah disebutkan dalam teks dan dicantumkan dalam Daftar Pustaka di bagian akhir skripsi ini.

Dengan ini saya melimpahkan hak cipta dari karya tulis saya kepada Institut Pertanian Bogor.

Bogor, April 2018

Raden Asri Ramadhina Fitriani
NIM G64154007

ABSTRAK

RADEN ASRI RAMADHINA FITRIANI. Instrumentasi Kode Program Secara Otomatis Untuk *Path Testing*. Dibimbing oleh IRMAN HERMADI.

Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan. *Path testing* merupakan salah satu metode pengujian struktural yang menggunakan kode program untuk menemukan semua jalur yang mungkin dapat dilalui ketika program tersebut dijalankan dan dapat digunakan untuk merancang data uji. Untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Pengujian perangkat lunak dapat menggunakan hampir 60% dari total biaya pengembangan perangkat lunak. Mengotomasi bagian dari pengujian akan membuat proses ini menjadi lebih cepat dan mengurangi kerawanan akan kesalahan. Pada penelitian ini, akan dibangun sebuah aplikasi untuk membangkitkan kemungkinan jalur-jalur dari sebuah program yang dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur yang dilalui program ketika dijalankan dengan masukan data uji tertentu, maka sistem ini juga akan melakukan instrumentasi kode program secara otomatis. Program yang akan diuji dalam penelitian ini adalah program yang dibangun dengan menggunakan bahasa *Matlab*. Dalam pengembangannya, aplikasi ini akan dibangun dengan menggunakan bahasa pemrograman C#.

Kata kunci: *Path Testing*, *Control Flow Graph*, Instrumentasi, *Matlab*

ABSTRACT

RADEN ASRI RAMADHINA FITRIANI. Automatic Source Code Instrumentation for Path Testing. Supervised by IRMAN HERMADI.

Testing is series of processes designed to ensure a software does what it's supposed to do. path testing is one of the structural testing methods that use source code to find all possible paths that can be passed and can be used to design test data. To test the complex software as a whole will take a long time and require a lot of human resources. Software testing can use nearly 60% of total software development costs. Automating part of the test will make the process more faster and reduce the vulnerability error. In this study, an application will be built to generate possible path from a program that can be used as a base for generating test data and that test data used for testing can represent all possibilities. To monitor the path the program passes when it is run with certain test data input, then this system as well will do automatic instrumentation source code. The program to be tested in this research is a program built using Matlab language. In its development, this application will be built using C# programming language.

Keywords: Path Testing, Control Flow Graph, Instrumentation, Matlab

**INSTRUMENTASI KODE PROGRAM SECARA OTOMATIS
UNTUK *PATH TESTING***

RADEN ASRI RAMADHINA FITRIANI

Skripsi
sebagai salah satu syarat untuk memperoleh gelar
Sarjana Komputer
pada
Departemen Ilmu Komputer

**DEPARTEMEN ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
INSTITUT PERTANIAN BOGOR
BOGOR
2018**

Penguji:

1 Prof Dr Ir Agus Buono, MSi MKom

2 Dr Wisnu Ananta Kusuma, ST MT

Judul Skripsi: Instrumentasi Kode Program Secara Otomatis untuk *Path Testing*
Nama : Raden Asri Ramadhina Fitriani
NIM : G64154007

Disetujui oleh

Irman Hermadi, Skom MS PhD
Pembimbing I

Diketahui oleh

Prof Dr Ir Agus Buono, MSi MKom
Ketua Departemen

Tanggal Lulus:

PRAKATA

Puji dan syukur penulis panjatkan kepada Allah *Subhanahu wa ta'ala* atas segala rahmat-Nya sehingga penulis dapat menyelesaikan skripsi ini dengan baik. Tema yang dipilih dalam penelitian yang dilaksanakan sejak bulan Juli 2017 ini ialah *computer scientist*, dengan judul Instrumentasi Kode Program Secara Otomatis untuk *Path Testing*.

Penulis ingin mengucapkan terima kasih sebanyak-banyaknya kepada:

- 1 Bapak Raden Achmad Mulyadi dan Ibu Tini Hertini sebagai orang tua penulis, serta Raden Indra Yuga Pratama dan Raden Agung Yuga Dwitama sebagai kakak-kakak penulis atas semangat, doa, dan dukungannya selama ini.
- 2 Bapak Irman Hermadi, Skom MS PhD selaku dosen pembimbing atas segala ilmu, nasihat, arahan, waktu dan kesabarannya dalam membimbing penulis untuk menyelesaikan skripsi ini.
- 3 Bapak Prof Dr Ir Agus Buono, MSi MKom dan Bapak Dr Wisnu Ananta Kusuma, ST MT selaku penguji yang telah memberikan saran dan masukan untuk menyelesaikan skripsi ini.
- 4 Seluruh dosen dan tenaga kependidikan Departemen Ilmu Komputer IPB
- 5 Ilham Tri Mulyawan, Aulia Afriza, dan Alin Nur Alifah sebagai orang-orang terdekat penulis yang telah membantu serta memberikan semangat dan dukungannya.
- 6 Teman-teman Program Sarjana Alih Jenis Ilmu Komputer IPB Angkatan 10 yang selalu berjuang bersama penulis selama menjalani perkuliahan.

Semoga segala bantuan, bimbingan, motivasi, dan dukungan yang telah diberikan kepada penulis senantiasa dibalas oleh Allah *Subhanahu wa ta'ala*.

Penulis menyadari bahwa skripsi ini masih banyak kekurangannya. Semoga skripsi ini dapat bermanfaat bagi siapapun yang membutuhkannya.

Bogor, April 2018

Raden Asri Ramadhina Fitriani

DAFTAR ISI

DAFTAR TABEL	vi
DAFTAR GAMBAR	vi
DAFTAR LAMPIRAN	vi
PENDAHULUAN	1
Latar Belakang	1
Perumusan Masalah	2
Tujuan Penelitian	2
Ruang Lingkup Penelitian	2
Manfaat Penelitian	2
METODE	2
Analisis	2
Perancangan	3
Implementasi	6
Testing	6
HASIL DAN PEMBAHASAN	6
Analisis	6
Perancangan	7
Implementasi	9
Testing	17
SIMPULAN DAN SARAN	21
Simpulan	21
Saran	21
DAFTAR PUSTAKA	21
LAMPIRAN	23
RIWAYAT HIDUP	44

DAFTAR TABEL

1	Contoh program uji	7
2	Perbandingan <i>adjacency list</i> manual dan menggunakan aplikasi	17
3	Perbandingan waktu eksekusi secara manual dan menggunakan aplikasi	20

DAFTAR GAMBAR

1	Tahapan penelitian	2
2	Arsitektur sistem	3
3	Notasi <i>Control Flow Graph</i> (CFG)	5
4	Perancangan <i>Class Diagram</i>	8
5	Perancangan antarmuka <i>form awal</i>	8
6	Perancangan antarmuka hasil dari proses yang telah dilakukan	9
7	Kode Program tA2008	10
8	Potongan hasil penguraian kode program tA2008 ke dalam format XML	10
9	Hasil <i>nodes</i> yang terbentuk dari tA2008	11
10	Representasi <i>Object Graph</i> tA2008 dalam bentuk	12
11	Representasi tA2008 dalam bahasa dot	13
12	CFG tA2008	13
13	Hasil instrumentasi tA2008	15
14	Hasil eksekusi kode program tA2008 yang sudah diinstrumentasi	15
15	Tampilan halaman awal aplikasi	15
16	Tampilan hasil pembangkitan	16

DAFTAR LAMPIRAN

1	Kode program mmA2008	23
2	Hasil instrumentasi mmA2008	23
3	CFG dan <i>Cyclomatic Complexity</i> mmA2008	24
4	Semua kemungkinan jalur mmA2008	24
5	Kode program iA2008	24
6	Hasil instrumentasi iA2008	25
7	CFG dan <i>Cyclomatic Complexity</i> iA2008	25
8	Semua kemungkinan jalur iA2008	25
9	Kode program binA2008	26
10	Hasil instrumentasi binA2008	26
11	CFG dan <i>Cyclomatic Complexity</i> binA2008	27
12	Semua kemungkinan jalur binA2008	27
13	Kode program bubA2008	28
14	Hasil instrumentasi bubA2008	28
15	CFG dan <i>Cyclomatic Complexity</i> bubA2008	29

16	Semua kemungkinan jalur bubA2008	29
17	Kode program gA2008	29
18	Hasil instrumentasi gA2008	30
19	CFG dan <i>Cyclomatic Complexity</i> gA2008	30
20	Semua kemungkinan jalur gA2008	31
21	Kode program eB2002	31
22	Hasil instrumentasi eB2002	32
23	CFG dan <i>Cyclomatic Complexity</i> eB2002	33
24	Semua kemungkinan jalur eB2002	34
25	Kode program qB2002	35
26	Hasil instrumentasi qB2002	35
27	CFG dan <i>Cyclomatic Complexity</i> qB2002	36
28	Semua kemungkinan jalur qB2002	36
29	Kode program fG2011	37
30	Hasil instrumentasi fG2011	38
31	CFG dan <i>Cyclomatic Complexity</i> fG2011	40
32	Semua kemungkinan jalur fG2011	41
33	Kode program fmH2015	41
34	Hasil instrumentasi fmH2015	42
35	CFG dan <i>Cyclomatic Complexity</i> fmH2015	42
36	Semua kemungkinan jalur fmH2015	43

PENDAHULUAN

Latar Belakang

Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan. Proses ini bertujuan untuk menemukan kesalahan pada perangkat lunak. Saat pengujian, bisa saja tidak ditemukan kesalahan pada hasil pengujian. Hal ini dapat terjadi karena perangkat lunak yang sudah berkualitas tinggi atau karena proses pengujiannya berkualitas rendah. (Myers *et al.* 2012).

Teknik pengujian secara umum dibagi menjadi 2 kategori diantaranya *black box testing* dan *white box testing*. *Black box testing* bertujuan untuk memeriksa fungsional dari perangkat lunak apakah output sudah sesuai dengan yang ditentukan. Sedangkan *white box testing* atau biasa disebut dengan pengujian struktural merupakan pemeriksaan struktur dan alur logika suatu proses. *Path testing* merupakan salah satu metode pengujian struktural yang menggunakan kode program untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Basu 2015). Untuk melakukan monitoring jalur mana yang diambil oleh sebuah masukan pada saat eksekusi program, maka diperlukan penanda yang dapat memberikan informasi cabang mana yang dilalui. Proses menyisipkan tanda tersebut disebut instrumentasi. Biasanya tanda tersebut disisipkan tepat sebelum atau sesudah sebuah percabangan (Tikir dan Hollingsworth 2011).

Idealnya, pengujian dilakukan untuk semua kemungkinan dari perangkat lunak. Tetapi untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Kumar dan Mishra (2016) mengatakan bahwa pengujian perangkat lunak menggunakan hampir 60% dari total biaya pengembangan perangkat lunak. Jika proses pengujian perangkat lunak dapat dilakukan secara otomatis, maka hal ini dapat mengurangi biaya pengembangan secara signifikan.

Hermadi (2015) melakukan penelitian membangkitkan data uji untuk *path testing* menggunakan algoritma genetika. Dalam penelitian tersebut, Hermadi membangkitkan *Control Flow Graph* (CFG) dan instrumentasi secara manual sehingga membutuhkan banyak waktu dan rawan akan kesalahan ketika program sudah semakin besar. Mengotomasi hal tersebut diharapkan dapat membuat *path testing* menjadi lebih cepat dan dapat mengurangi kerawanan akan kesalahan.

Pada penelitian ini, akan dibangun sebuah perangkat lunak untuk membangkitkan semua kemungkinan jalur dari sebuah program. Jalur-jalur ini dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur mana yang dilalui ketika diberikan masukan data uji, maka sistem ini juga akan melakukan penyisipan *tag-tag* sebagai instrumentasi ke dalam kode program secara otomatis.

Perumusan Masalah

Berdasarkan latar belakang di atas dapat dirumuskan masalahnya adalah bagaimana membangun sebuah aplikasi untuk melakukan instrumentasi secara otomatis untuk pengujian jalur dan dapat dimanfaatkan untuk *re-engineering* perangkat lunak.

Tujuan Penelitian

Penelitian ini bertujuan untuk membangun sebuah aplikasi yang dapat digunakan untuk membangkitkan CFG dan melakukan instrumentasi kode program secara otomatis.

Ruang Lingkup Penelitian

Bahasa pemrograman yang diakomodasi adalah *Matlab* dan model diagram yang dibangkitkan adalah CFG.

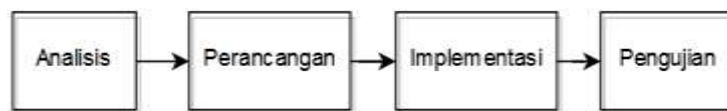
Manfaat Penelitian

Hasil penelitian diharapkan dapat membantu pengembang dan penguji aplikasi untuk:

1. Menyisipkan *tag-tag* sebagai instrumentasi program ke dalam kode program secara otomatis sehingga proses tersebut dapat dilakukan dengan lebih cepat.
2. Membangkitkan jalur-jalur dasar yang dapat digunakan sebagai dasar untuk pembangkitan data uji.
3. Membangkitkan diagram CFG yang dapat memudahkan pengembang dalam memahami struktur dan alur dari suatu program yang dapat dimanfaatkan ketika akan melakukan *re-engineering* perangkat lunak

METODE

Penelitian yang dilakukan terbagi menjadi beberapa tahapan proses. Gambar 1 menunjukkan tahapan proses tersebut.



Gambar 1 Tahapan penelitian

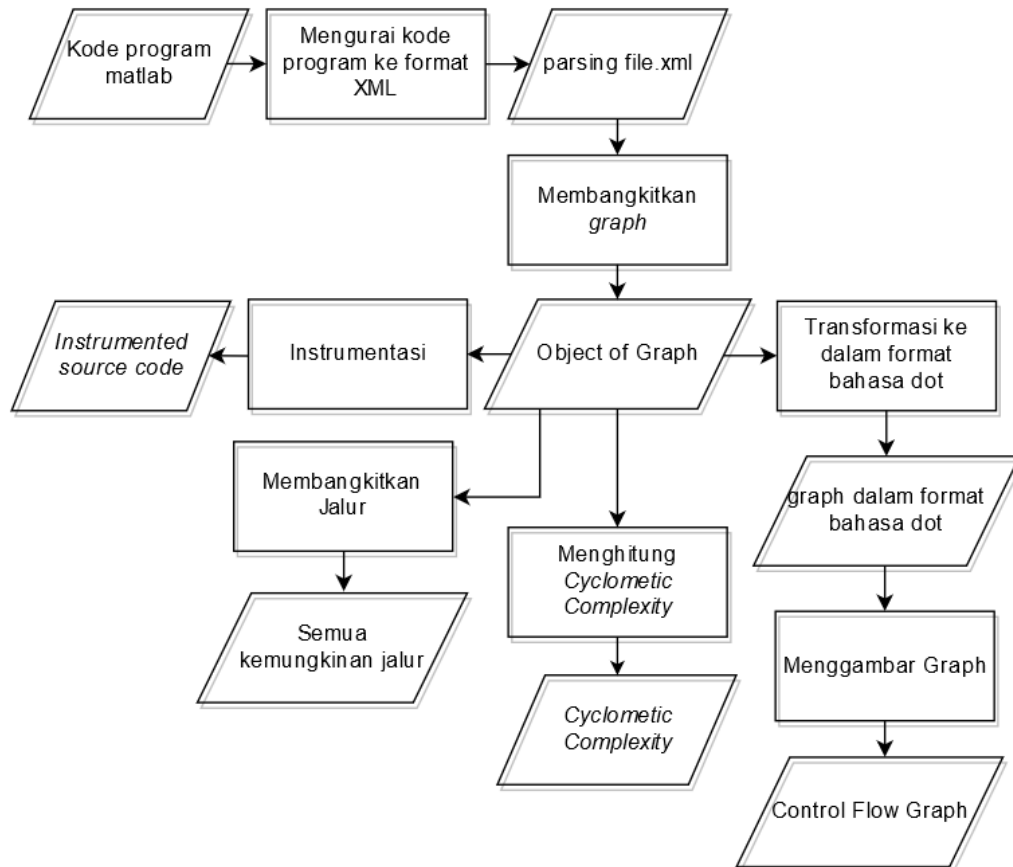
Analisis

Pada tahap ini dimulai dari membaca literatur terkait dan mendefinisikan kebutuhan dari aplikasi yang akan dibangun. Selain itu, pada tahapan ini juga dilakukan pengumpulan beberapa contoh program yang akan digunakan dalam

penelitian. Contoh program yang akan digunakan pada penelitian ini diperoleh dari penelitian yang dilakukan oleh Hermadi (2015).

Perancangan

Pada tahap ini ditentukan bagaimana perangkat lunak akan dibangun. Ilustrasi arsitektur sistem dapat dilihat pada Gambar 2.



Gambar 2 Arsitektur sistem

Kode Program

Kode program matlab akan dibaca sebagai inputan. *Matlab* merupakan singkatan dari *MATrix LABoratory*. Seperti bahasa pemrograman lainnya, matlab memiliki beberapa kontrol struktur. Kontrol struktur adalah perintah dalam bahasa pemrograman yang digunakan dalam pengambilan keputusan. *Matlab* memiliki empat kontrol struktur, yaitu *IF-ELSE-END*, *SWITCH-CASE*, *FOR*, dan *WHILE* (Houcque 2005).

Mengurai Kode Program ke Format XML

Penguraian kode program matlab dilakukan dengan menggunakan *library MATLAB-PARSER*. Lalu kode program tersebut diurai menjadi *file* dengan format XML menggunakan *library MATLAB-PARSER* yang dibangun oleh Suffos (2015).

Extensible Markup Language (XML) adalah bahasa yang dapat mendeskripsikan sebuah dokumen. XML memiliki banyak bagian yang tidak

memiliki struktur yang pasti. XML terdiri atas dua bagian utama, yaitu elemen dan atribut. Elemen yang dapat disebut sebagai *node* merupakan bagian penting yang dapat menggambarkan struktur dari XML. Sedangkan atribut merupakan bagian yang dapat digunakan sebagai informasi tambahan dari setiap elemen (Hartwell 2017).

Membangkitkan *Graph*

Salah satu cara untuk membaca dan menulis dokumen XML pada *framework* .NET dan C# yaitu dengan menggunakan kelas *XMLDocument* yang terdapat dalam *namespace System.XML*. Setiap elemen XML yang merupakan kontrol struktur pada program akan menjadi *node* baru. Setiap *node* berisi informasi nomor baris dan nomor kolom yang akan digunakan untuk melakukan instrumentasi. Sehingga terbentuklah sebuah objek *graph* yang terdiri dari sekumpulan *node* dan *edge*.

Membangkitkan Jalur

Path testing merupakan salah satu metode pengujian struktural yang menggunakan kode sumber atau program (*source code*) untuk menemukan semua jalur yang mungkin dapat dilalui program dan dapat digunakan untuk merancang data uji. Metode ini memastikan semua kemungkinan jalur dijalankan setidaknya satu kali (Basu 2015).

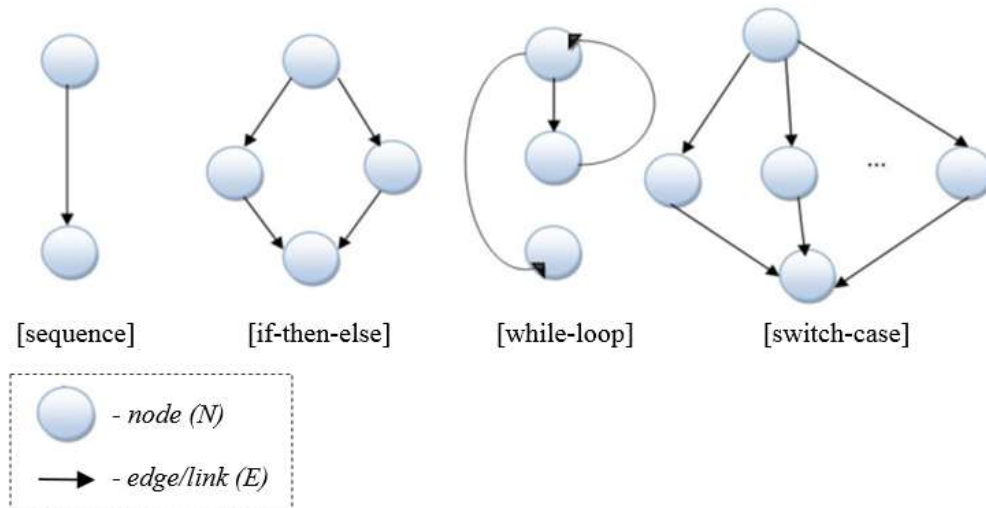
Jalur dibentuk dengan cara menelusuri objek *graph* yang sudah dibentuk sebelumnya. Jika *edge* memiliki tipe *true* atau *false*, maka jalur yang dibangkitkan akan ditambahkan informasi cabang yang dilalui. (T) ketika melalui *edge* yang memiliki tipe *true*, dan (F) ketika melalui *edge* yang memiliki tipe *false*. Setiap *edge* memiliki atribut *isVisited* yang digunakan untuk menandai apakah garis penghubung tersebut sudah dilalui atau belum. Jalur yang dibentuk ketika melalui perintah pengulangan seperti FOR dan WHILE akan dibatasi hanya satu kali pengulangan.

Transformasi ke Dalam Format Bahasa *Dot*

Graph yang sudah terbentuk akan ditransformasikan ke dalam bentuk format bahasa pemrograman *dot*. Bahasa *dot* adalah bahasa yang digunakan untuk menggambar *graph* berarah. Bahasa ini dapat mendeskripsikan 3 macam objek, yaitu *graph*, *nodes*, dan *edges* (Ganser *et al* 2015).

Memvisualisasikan *Graph* dalam bentuk CFG

Control Flow Graph (CFG) adalah *graph* berarah yang merepresentasikan aliran dari sebuah program. Setiap CFG terdiri dari *nodes* dan *edges*. *Nodes* merepresentasikan perintah. Sedangkan *edges* merepresentasikan transfer kontrol antar *nodes* (Watson dan McCabe 1996). Notasi dari CFG dapat dilihat pada Gambar 3.



Gambar 3 Notasi *Control Flow Graph* (CFG)

Setelah *file* dengan format bahasa *dot* terbentuk, CFG akan divisualisasikan dengan menggunakan *library Graphviz*. *Graphviz* merupakan perangkat lunak *open source* untuk visualisasi grafik. *Graphviz* memiliki banyak fitur berguna untuk menggambar diagram yang konkret karena terdapat pilihan warna, *font*, tata letak, jenis garis, dan bentuk (Ellson *et al* 2003).

Menghitung *Cyclomatic Complexity*

Cyclomatic complexity merupakan suatu sistem pengukuran yang ditemukan oleh Watson dan McCabe (1996) untuk menentukan banyaknya *independent path* dan menunjukkan tingkat kompleksitas dari suatu program. *Independent path* adalah jalur yang melintas dalam program yang sekurang-kurangnya terdapat kondisi baru. Perhitungan *Cyclomatic Complexity* dapat dilihat pada persamaan berikut:

$$V(G) = E - N + 2$$

dimana, E menunjukkan jumlah *edges* dan N menunjukkan jumlah *nodes*.

Instrumentasi

Setelah jalur terbentuk, dilakukan juga proses instrumentasi. Instrumentasi merupakan sebuah proses menyisipkan sebuah penanda (*tag*) di awal atau di akhir setiap blok kode. Penyisipan dapat dilakukan sebelum atau sesudah kondisi terpenuhi atau tidak. Dalam *path testing*, penanda ini dapat digunakan untuk memonitor jalur yang dilalui program ketika dijalankan dengan masukan data uji tertentu (Arkeman *et al.* 2014).

Instrumentasi akan dilakukan dengan cara menambahkan dulu variabel keluaran bernama *traversedPath*. Variabel ini digunakan untuk menyimpan informasi *node* mana saja yang dilalui ketika diberikan inputan dengan nilai tertentu. Lalu setiap sebelum dan sesudah *node* percabangan, dilakukan penyisipan kode program berupa perintah untuk memasukkan nilai *node* yang dilalui. Sehingga ketika program tersebut dijalankan, akan menghasilkan keluaran tambahan bernama *traversedPath*.

Implementasi

Tahapan ini adalah melakukan implementasi dari tahap sebelumnya ke dalam bentuk aplikasi web. Aplikasi ini akan dibangun dengan menggunakan bahasa pemrograman C# dan menggunakan *IDE Microsoft Visual Studio Ultimate 2013*.

Setelah file dengan format bahasa dot terbentuk, CFG divisualisasikan dengan menggunakan *library Graphviz.Net*. *Graphviz.Net* adalah pembungkus C# untuk generator grafik *Graphviz* yang dibangun oleh Dixon (2013). Keluaran yang dikembalikan ketika mengeksekusi *Graphviz.Net* berbentuk *byte* dalam *array* sehingga dapat diolah kembali sesuai dengan kebutuhan. *Graphviz* merupakan *library* yang dapat digunakan untuk divisualisasi jalur ke dalam bentuk *graph* berarah (Gansner 2015).

Testing

Tahapan ini adalah melakukan evaluasi dari tahapan implementasi. Evaluasi dibagi menjadi dua bagian, yaitu uji validasi dan uji efisiensi. Uji validasi dilakukan dengan cara membandingkan hasil yang ada pada penelitian sebelumnya dengan hasil yang dikeluarkan oleh aplikasi. Pada penelitian sebelumnya, *graph* yang dibangun adalah *graph* yang hanya menggambarkan notasi percabangan dan tanpa penomoran *node*. Agar dapat dibandingkan dengan hasil yang dikeluarkan oleh aplikasi, *graph* yang ada pada penelitian sebelumnya direpresentasikan ke dalam bentuk *adjacency list* terlebih dahulu secara manual.

Uji efisiensi dilakukan dengan membandingkan waktu eksekusi yang dilakukan secara manual dengan waktu eksekusi oleh aplikasi. Pengujian manual akan dilakukan dengan meminta satu atau dua orang yang sudah memiliki pengalaman dalam pemrograman sebagai sampel untuk melihat berapa lama waktu yang dibutuhkan untuk membangkitkan CFG, membangkitkan semua kemungkinan jalur, menghitung *cyclomatic complexity*, dan melakukan instrumentasi. Dalam pengujian efisiensi, waktu yang diperoleh dari pengujian melalui aplikasi adalah waktu eksekusi dari aplikasi yang dijalankan di dalam local komputer tanpa menggunakan koneksi internet.

HASIL DAN PEMBAHASAN

Analisis

Pada penelitian ini, akan dibangun sebuah perangkat lunak untuk membangkitkan kemungkinan jalur dari sebuah program. Jalur-jalur ini dapat dijadikan dasar untuk membangkitkan data uji agar data uji yang digunakan untuk pengujian dapat mewakili semua kemungkinan. Untuk memonitor jalur mana yang dilalui ketika diberikan masukan data uji, maka sistem ini juga akan melakukan penyisipan *tag-tag* sebagai instrumentasi ke dalam kode program secara otomatis.

Sebelumnya sudah terdapat beberapa program yang dapat membangkitkan CFG seperti *Eclipse Control Flow Graph Generator* tetapi *library* tersebut hanya dapat digunakan di eclipse dan hanya membangkitkan CFG dari kode program java (Alimucaj 2009).

Data yang digunakan dalam penelitian ini diperoleh dari penelitian yang dilakukan oleh Hermadi (2015). Terdapat 10 contoh program yang akan digunakan pada penelitian ini dengan tingkat kompleksitas yang beragam. Contoh program yang akan digunakan dapat dilihat pada Tabel 1.

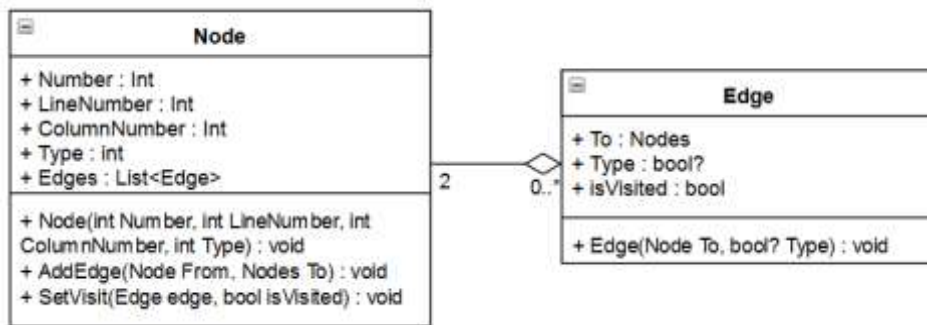
Tabel 1 Contoh program uji

No	Program Uji	Nama	Deskripsi
1	Triangle Ahmed	tA2008	Menentukan tipe dari segitiga apakah termasuk <i>equilateral</i> , <i>isosceles</i> , <i>scalene</i> , atau <i>not triangle</i>
2	Minimaxi Ahmed	mmA2008	Menentukan nilai minimal dan maksimal dari inputan berupa bilangan dalam <i>array</i>
3	Insertion Ahmed	iA2008	Mengurutkan bilangan dalam <i>array</i> menggunakan metode <i>insertion sort</i>
4	Binary Ahmed	binA2008	mencari indeks sebuah bilangan dalam <i>array</i> dengan mengembalikan indeks jika ditemukan dan tidak jika tidak ditemukan.
5	Bubble Ahmed	bubA2008	Mengurutkan bilangan dalam <i>array</i> menggunakan metode <i>bubble sort</i>
6	Expint Bueno	eB2002	Fungsi eksponensial yang dapat memproses bilangan <i>integer</i> dan <i>float</i>
7	Gcd Ahmed	gA2008	Menghitung <i>GCD</i> atau pembagi dua bilangan terbesar
8	Quotient Bueno	qB2002	Menghitung hasil bagi dan sisa hasil bagi dari dua buah bilangan bulat positif
9	Flex Gong	fG2011	Sebuah utilitas <i>unix</i> yang diambil dari situs GNU
10	Fitness Minimaxi Hermadi	fmH2015	Menghitung fungsi <i>fitness</i> dari fungsi minimaxiAhmed2008

Perancangan

Perancangan Class Diagram

Class diagram dibangun untuk menggambarkan struktur sistem dari segi pendefinisian *class* dan hubungan antar *class*. Perancangan *class diagram* dapat dilihat pada Gambar 4.

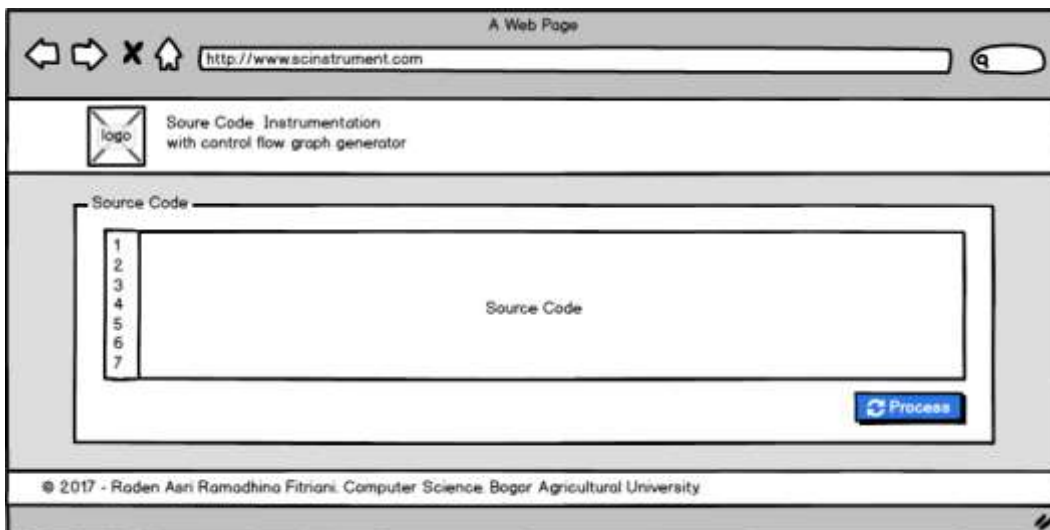


Gambar 4 Perancangan *Class Diagram*

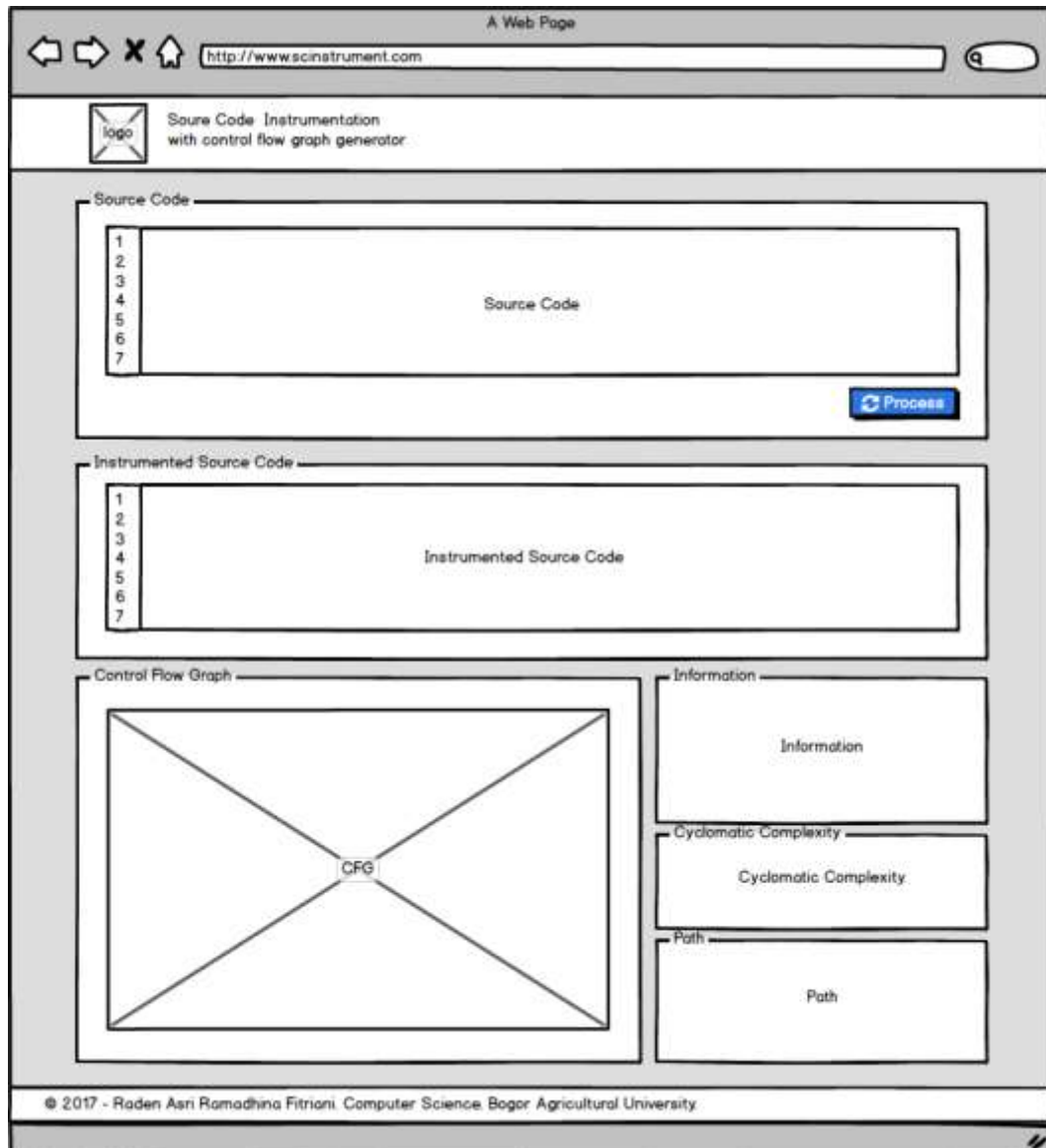
Dalam sebuah *class node* terdapat informasi nomor *node*, nomor baris dan nomor kolom dari kode program, dan tipe dari perintah tersebut apakah termasuk percabangan, pengulangan, perintah biasa, atau akhir dari sebuah perintah. Selain itu, terdapat list *edge* yang berisi *node* tujuan dan tipe dari *edge* yang digunakan jika terdapat percabangan *true*, *false*, atau hanya garis penghubung biasa.

Perancangan Antarmuka

Perancangan antarmuka meliputi perancangan antarmuka *form* untuk pengguna memasukkan kode program yang akan di proses dan antarmuka hasil dari proses yang telah dilakukan oleh aplikasi. Perancangan antarmuka *form* awal yang digunakan untuk memasukkan kode program yang akan di proses dapat dilihat pada Gambar 5 dan perancangan antarmuka hasil dari proses yang telah dilakukan dapat dilihat pada Gambar 6



Gambar 5 Perancangan antarmuka *form awal*



Gambar 6 Perancangan antarmuka hasil dari proses yang telah dilakukan

Implementasi

Aplikasi dibangun dengan menggunakan bahasa pemrograman C# dan menggunakan *IDE Microsoft Visual Studio Ultimate 2013*.

Sebagai contoh, kode program yang digunakan adalah tA2008. Pada kode tA2008 terdapat perintah *IF-THEN-ELSE* bersarang sebanyak tiga tingkat.

Kode Program

Kode program tA2008 dapat dilihat pada Gambar 7. Program ini digunakan untuk mencari jenis dari segitiga jika diketahui panjang dari setiap sisinya.

```

1 function type = triangle(sideLengths)
2     A = sideLengths(1); % First side
3     B = sideLengths(2); % Second side
4     C = sideLengths(3); % Third side
5     if ((A+B > C) && (B+C > A) && (C+A > B))
6         if ((A ~= B) && (B ~= C) && (C ~= A))
7             type = 'Scalene';
8         else
9             if ((A == B) && (B == C)) || ((B == C) && (C == A)) || ((C == A) && (A == B))
10                 type = 'Isosceles';
11             else
12                 type = 'Equilateral';
13             end
14         end
15     else
16         type = 'Not a triangle';
17     end
18 end

```

Gambar 7 Kode Program tA2008

Mengurai Kode Program ke Format XML

Penguraian kode program matlab dilakukan dengan menggunakan *library MATLAB-PARSER*. Kode program yang diinputkan harus sudah dipastikan dapat dijalankan jika di *compile*. Ketika terdapat kesalahan pada kode program, *library* ini akan mengembalikan pesan *error*. Gambar 8 menunjukkan potongan hasil penguraian kode program tA2008 ke dalam format XML. Potongan kode XML yang terlihat pada Gambar 8 menunjukkan hasil penguraian dari kode program pada baris ke 6 sampai baris ke 7.

```

278 <IfPart.Statements>
279 <If Line="6" Column="2" Text="if">
280 <IfPart>
281 <IfPart Line="6" Column="2" Text="if">
282 <IfPart.Expression>
283 <ShortAnd Line="6" Column="27" Text="&";&";>
284 </IfPart.Expression>
285 </IfPart>
286 <IfPart.Statements>
287 <Assignment Line="7" Column="8" Text="=">
288 <Assignment.LValue>
289 <Var Line="7" Column="3" Text="">
290 <Var.Name>
291 <Name Line="7" Column="3" Text="">
292 <Name.Ids>
293 <Id Line="7" Column="3" Text="type" />
294 </Name.Ids>
295 </Name>
296 </Var.Name>
297 </Var>
298 </Assignment.LValue>
299 <Assignment.Value>
300 <String Line="7" Column="10" Text="'Scalene'" />
301 </Assignment.Value>
302 <Assignment.Terminator>
303 <NoPrint Line="7" Column="19" Text=";" />
304 </Assignment.Terminator>
305 </Assignment>
306 </IfPart.Statements>
307 </IfPart>
308 </If.IfPart>
309 <If.ElsePart>
310 <ElsePart Line="8" Column="2" Text="else">

```

Gambar 8 Potongan hasil penguraian kode program tA2008 ke dalam format XML

Ketika ditemukan perintah *IF* maka akan dibentuk sebuah elemen `<if></if>`. Lalu untuk bagian memenuhi kondisi *IF* akan disimpan di dalam elemen `<If.IfPart></If.IfPart>`. Ekspresi dari kondisi *IF* akan disimpan di dalam elemen `<IfPart.Expression></IfPart.Expression>`. Perintah yang akan dilakukan ketika memenuhi kondisi *IF* akan disimpan dalam elemen `<IfPart.Statements></IfPart.Statements>`. Sedangkan untuk bagian yang tidak memenuhi kondisi *IF* atau bagian *ELSE* akan disimpan di dalam elemen `<If.ElsePart></If.ElsePart>`.

Membangkitkan *Graph*

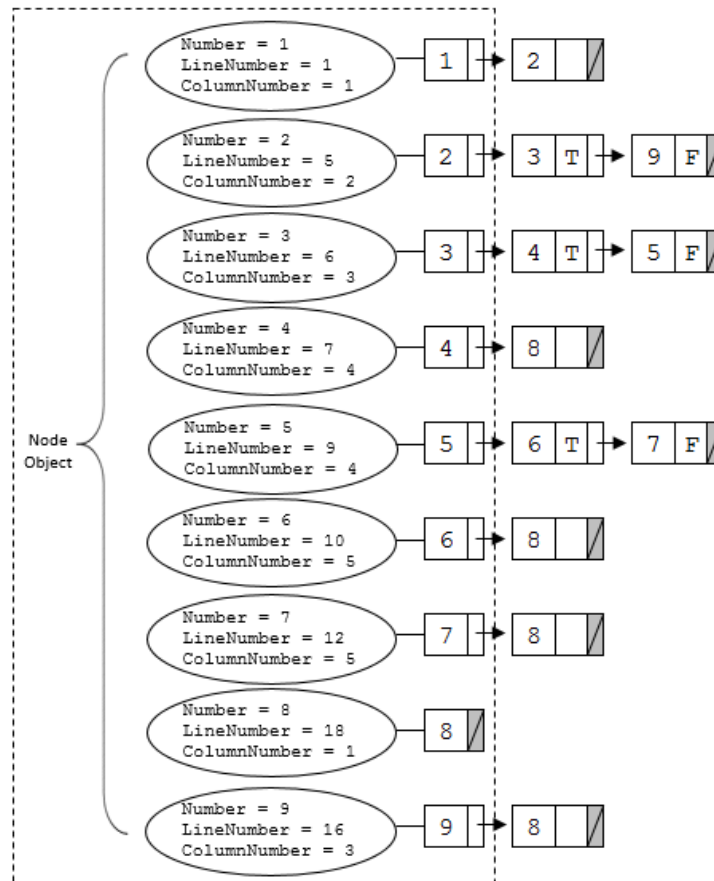
Salah satu cara untuk membaca dan menulis dokumen XML pada *framework* .NET dan C# yaitu dengan menggunakan kelas *XMLDocument* yang terdapat dalam *namespace* *System.XML*. Setiap elemen XML yang merupakan kontrol struktur pada program akan menjadi *node* baru. Setiap *node* berisi informasi nomor baris dan nomor kolom yang akan digunakan untuk melakukan instrumentasi. Setiap *node* juga dapat memiliki *edge* yang berisi informasi *node* tujuan dan tipe dari garis penghubung itu sendiri. Terdapat tiga macam tipe pada *edge* yaitu, *null*, *true*, dan *false*. *True* dan *false* digunakan jika *node* asal merupakan percabangan.

Hasil *node* yang dibentuk dari kode program tA2008 dapat dilihat pada Gambar 9. *Node* 1 dibentuk pada awal kode program sebagai inisialisasi. *Node* ditambahkan ketika bertemu dengan perintah yang termasuk ke dalam kontrol struktur seperti *IF-ELSE-END*, *SWITCH-CASE*, *FOR*, dan *WHILE*. Seperti dapat dilihat pada baris kode ke 5, terdapat perintah *IF* sehingga dibentuk *node* baru yaitu *node* 2.

1	Node 1	function type = triangle(sideLengths)
2		A = sideLengths(1); % First side
3		B = sideLengths(2); % Second side
4		C = sideLengths(3); % Third side
5	Node 2	if ((A+B > C) && (B+C > A) && (C+A > B))
6	Node 3	if ((A ~= B) && (B ~= C) && (C ~= A))
7	Node 4	type = 'Scalene';
8		else
9	Node 5	if (((A == B) && (B ~= C)) ((B == C) && (C ~= A)) ((C == A) && (A ~= B)))
10	Node 6	type = 'Isosceles';
11		else
12	Node 7	type = 'Equilateral';
13		end
14		end
15		else
16	Node 9	type = 'Not a triangle';
17	Node 8	end

Gambar 9 Hasil *nodes* yang terbentuk dari tA2008

Representasi objek dari kelas *graph* yang terbentuk dari kode program tA2008 dapat dilihat pada Gambar 10. *Graph* disimpan ke dalam struktur data *adjacency list* dari objek *node* yang dihubungkan oleh objek *edge*. Terbentuk 9 buah *nodes* dan 11 buah *edges* yang menghubungkan antar *nodes* tersebut.



Gambar 10 Representasi *Object Graph* tA2008 dalam bentuk

Membangkitkan Jalur

Jalur dibentuk dengan cara menelusuri objek *graph* yang sudah dibentuk sebelumnya. Dari objek *graph* yang terbentuk dari kode program tA2008, terlihat jalur yang terbentuk dimulai dari *node* 1, lalu terhubung ke *node* 2. *Node* 2 merupakan *node* yang memiliki 2 percabangan yaitu cabang menuju *node* 3 yang dihubungkan oleh *edge* yang bernilai *true* dan cabang menuju *node* 9 yang dihubungkan oleh *edge* yang bernilai *false*. Sehingga akan menjadi jalur yang berbeda.

Berikut merupakan semua kemungkinan jalur yang akan yang dapat dijadikan sebagai dasar dalam pembangkitan data uji.

1. 1 2 (T) 3 (T) 4 8
2. 1 2 (T) 3 (F) 5 (T) 6 8
3. 1 2 (T) 3 (F) 5 (F) 7 8
4. 1 2 (F) 9 8

Transformasi ke Dalam Format Bahasa *Dot*

Transformasi ke dalam format bahasa *dot* dilakukan dengan cara menelusuri objek *graph* yang sudah dibangun sebelumnya. Yang didefinisikan dalam bahasa

dot adalah *edge* yang terdapat pada *graph* yang dibangun. Seperti yang dapat dilihat pada Gambar 11, jumlah baris sebanyak jumlah *edge* pada objek *graph* yang telah didefinisikan sebelumnya.

```

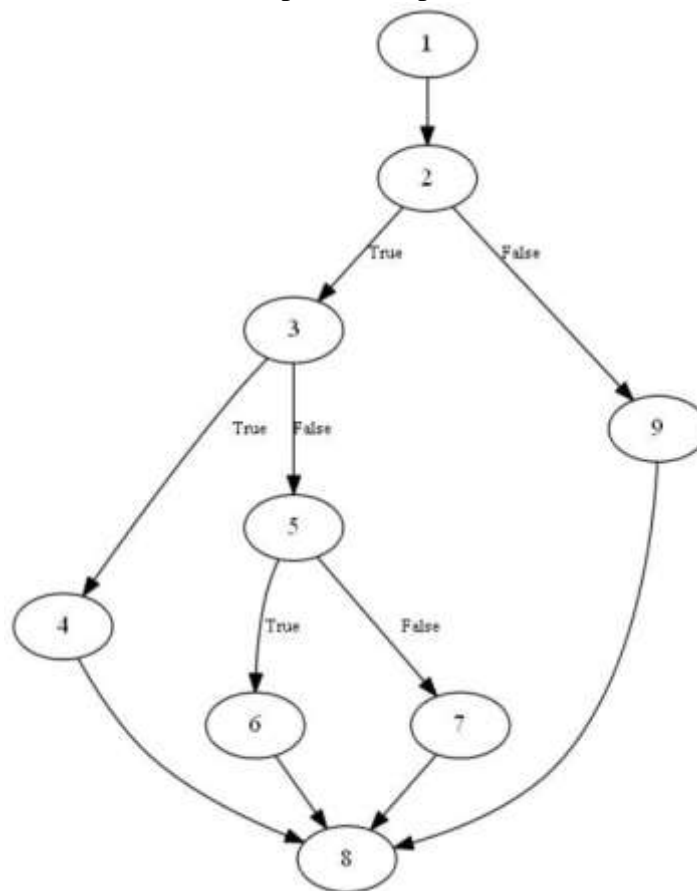
1 digraph G {
2 graph [label="" nodesep=0.8]
3 1->2;
4 2->3 [ label="True"  fontsize=10 ];
5 3->4 [ label="True"  fontsize=10 ];
6 3->5 [ label="False" fontsize=10 ];
7 5->6 [ label="True"  fontsize=10 ];
8 5->7 [ label="False" fontsize=10 ];
9 7->8;
10 6->8;
11 4->8;
12 2->9 [ label="False" fontsize=10 ];
13 9->8;
14 }

```

Gambar 11 Representasi tA2008 dalam bahasa dot

Memvisualisasikan *Graph*

Setelah file dengan format bahasa *dot* terbentuk, CFG divisualisasikan dengan menggunakan *library Graphviz.Net*. Hasil visualisasi bahasa *dot* kode program tA2008 ke dalam CFG dapat dilihat pada Gambar 12.



Gambar 12 CFG tA2008

Menghitung Cyclomatic Complexity

Cyclomatic complexity merupakan suatu sistem pengukuran yang menunjukkan banyaknya *independent path*. *Cyclomatic Complexity* dihitung dengan cara jumlah *edge* dikurangi dengan jumlah *node*, lalu ditambahkan dengan dua. Berdasarkan *graph* yang telah terbentuk dari kode program tA2008, dapat dilihat pada Gambar 10 bahwa jumlah *node* yang terbentuk adalah 9 dan jumlah *edge* yang terbentuk adalah 11. Sehingga hasil perhitungan *cyclomatic complexity* dapat dilihat pada persamaan dibawah ini.

$$\begin{aligned} \text{Nodes}(G) &= 9 \\ \text{Edges}(G) &= 11 \\ V(G) &= E - N + 2 \\ &= 4 \end{aligned}$$

Instrumentasi

Instrumentasi dilakukan dengan cara menambahkan dulu variabel keluaran bernama *traversedPath*. Variabel ini digunakan untuk menyimpan informasi *node* mana saja yang dilalui ketika diberikan inputan dengan nilai tertentu. Dan menyimpan informasi pilihan yang dilalui ketika ditemukan cabang yang terdapat pilihan *true* atau *false*.

Hasil kode program yang telah diinstrumentasi dapat dilihat pada Gambar 13. Sebelumnya, kode program tA2008 hanya mengembalikan keluaran satu variabel bernama *type* yaitu menunjukan jenis dari segitiga ketika diberikan panjang dari ketiga sisi segitiga. Setelah dilakukan instrumentasi, kode program tA2008 akan mengembalikan keluaran dengan variabel tambahan bernama *traversedPath*. Sehingga ketika program tersebut dijalankan dengan inputan tertentu akan menghasilkan keluaran nilai *traversedPath* dan *type* seperti yang dapat dilihat pada Gambar 14.

Sehingga ketika kode program hasil instrumentasi dijalankan dengan inputan tertentu akan menghasilkan keluaran variabel *traversedPath* dan *type* seperti yang dapat dilihat pada Gambar 14. Misalkan inputan adalah 3, 4, dan 4 akan menghasilkan *isosceles* dan dapat diketahui bagaimana cara menghasilkan keluaran tersebut dari *traversedPath*.

```

1 function [traversedPath,type] = triangle(sideLengths)
2     traversedPath = [];
3     traversedPath = [traversedPath '1 ' ];
4     A = sideLengths(1); % First side
5     B = sideLengths(2); % Second side
6     C = sideLengths(3); % Third side
7     % instrument Branch # 1
8     traversedPath = [traversedPath '2 ' ];
9     if ((A+B > C) && (B+C > A) && (C+A > B))
10         traversedPath = [traversedPath '(T) ' ];
11         % instrument Branch # 2
12         traversedPath = [traversedPath '3 ' ];
13         if ((A ~= B) && (B ~= C) && (C ~= A))
14             traversedPath = [traversedPath '(T) ' ];
15             traversedPath = [traversedPath '4 ' ];
16             type = 'Scalene';
17         else
18             traversedPath = [traversedPath '(F) ' ];

```

```

19 % instrument Branch # 3
20 traversedPath = [traversedPath '5 '];
21 if ((A == B) && (B ~= C)) || ((B == C) && (C ~= A))
22     || ((C == A) && (A ~= B))
23     traversedPath = [traversedPath '(T) '];
24     traversedPath = [traversedPath '6 '];
25     type = 'Isosceles';
26 else
27     traversedPath = [traversedPath '(F) '];
28     traversedPath = [traversedPath '7 '];
29     type = 'Equilateral';
30 end
31 else
32     traversedPath = [traversedPath '(F) '];
33     traversedPath = [traversedPath '9 '];
34     type = 'Not a triangle';
35 end
36 traversedPath = [traversedPath '8 '];
37 end

```

Gambar 13 Hasil instrumentasi tA2008

```

>> [traversedPath, type] = triangle([3,4,4])

traversedPath =

    '1 2 (T) 3 (F) 5 (T) 6 8 '

type =

    'Isosceles'

```

Gambar 14 Hasil eksekusi kode program tA2008 yang sudah diinstrumentasi

Implementasi Antarmuka

Halaman awal dari aplikasi dapat dilihat pada Gambar 15 dimana terdapat satu *text area* untuk memasukkan kode program yang akan di proses atau bisa juga melalui *file* dengan ekstensi *file Matlab (.m)*. Tampilan hasil pembangkitan dapat dilihat pada Gambar 16.



Gambar 15 Tampilan halaman awal aplikasi

Source Code

```

1  function type = triangle(sideLengths)
2
3      E = sideLengths(1) * 1/2 * side
4      E = sideLengths(2) * 1/2 * side
5      C = sideLengths(3) * 1/2 * side
6      if ((A > B) && (B > C) && (C > A))
7          if ((A == B) && (B == C) && (C == A))
8              type = "Equilateral";
9          else
10             type = "Isosceles";
11         end
12     else
13         type = "Not a triangle";
14     end
15 end

```

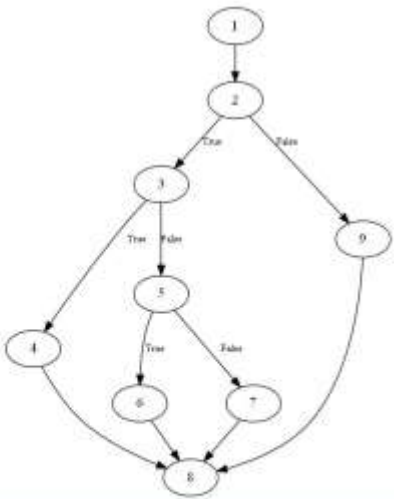
Instrumented Source Code

```

1  function [traversedPath, type] = triangle(sideLengths)
2
3      traversedPath = [];
4      E = sideLengths(1) * 1/2 * side
5      E = sideLengths(2) * 1/2 * side
6      C = sideLengths(3) * 1/2 * side
7      if ((A > B) && (B > C) && (C > A))
8          if ((A == B) && (B == C) && (C == A))
9              traversedPath = [traversedPath '2 '];
10             traversedPath = [traversedPath '3 '];
11             traversedPath = [traversedPath '4 '];
12             type = "Equilateral";
13         else
14             traversedPath = [traversedPath '5 '];
15             traversedPath = [traversedPath '6 '];
16             traversedPath = [traversedPath '7 '];
17             type = "Isosceles";
18         end
19     else
20         traversedPath = [traversedPath '8 '];
21         traversedPath = [traversedPath '9 '];
22         type = "Not a triangle";
23     end
24 end

```

Control Flow Graph



Information

```

function type = triangle(sideLengths)  node 1
E = sideLengths(1) * 1/2 * side
E = sideLengths(2) * 1/2 * side
C = sideLengths(3) * 1/2 * side
if ((A > B) && (B > C) && (C > A))  node 2
    if ((A == B) && (B == C) && (C == A))  node 3
        traversedPath = [traversedPath '2 '];  node 3
        traversedPath = [traversedPath '3 '];  node 3
        traversedPath = [traversedPath '4 '];  node 3
        type = "Equilateral";  node 3
    else  node 4
        traversedPath = [traversedPath '5 '];  node 4
        traversedPath = [traversedPath '6 '];  node 4
        traversedPath = [traversedPath '7 '];  node 4
        type = "Isosceles";  node 4
    end  node 5
else  node 6
    traversedPath = [traversedPath '8 '];  node 6
    traversedPath = [traversedPath '9 '];  node 6
    type = "Not a triangle";  node 6
end  node 7
end

```

Cyclomatic Complexity

$N(D) = 9$
 $E(G) = 11$
 $V(G) = E - N + 2$
 $= 2$

Path

```

1 1 2 (T) 3 (T) 4 8
2 1 2 (T) 3 (F) 5 (T) 6 8
3 1 2 (T) 3 (F) 5 (F) 7 8
4 1 2 (F) 9 8

```

© 2019 - Radeh Aul Hamadkha Pitaruk - Computer Science, Bogor Agricultural University

Source: Tump, 4.12.19

Gambar 16 Tampilan hasil pembangunan

Testing

Tahapan ini adalah melakukan evaluasi dari tahapan implementasi. Evaluasi dibagi menjadi dua bagian, yaitu uji validasi dan uji efisiensi. Uji validasi dilakukan dengan cara membandingkan hasil yang ada pada penelitian sebelumnya dengan hasil yang dikeluarkan oleh aplikasi. Pada penelitian sebelumnya, *graph* yang dibangun adalah *graph* yang hanya menggambarkan notasi percabangan. Agar dapat dibandingkan dengan hasil yang dikeluarkan oleh aplikasi, *graph* yang ada pada penelitian sebelumnya direpresentasikan ke dalam bentuk *adjacency list* terlebih dahulu secara manual. Tabel 2 menunjukkan *adjacency list* yang dibangun berdasarkan pada penelitian sebelumnya dan *adjacency list* yang dibangun menggunakan aplikasi.

Dari 10 program uji, bentuk *graph* yang terbentuk jika divisualisasikan dalam bentuk CFG sama. Perbedaan hanya terdapat pada label penomoran beberapa *node*. Seperti pada contoh program tA2008, *node* 4 dengan 5 dan 9 dengan 8 tertukar di *graph* yang dibangun menggunakan aplikasi.

Tabel 2 Perbandingan *adjacency list* manual dan menggunakan aplikasi

No	Nama Program	Adjacency List Manual	Adjacency List Aplikasi
1	tA2008	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 5 • 3 -> 4 -> 6 • 5 -> 9 • 4 -> 7 -> 8 • 6 -> 9 • 7 -> 9 • 9 • 8 -> 9 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 9 • 3 -> 4 -> 5 • 4 -> 8 • 5 -> 6 -> 7 • 6 -> 8 • 7 -> 8 • 8 • 9 -> 8
2	mmA2008	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 8 • 3 -> 4 -> 5 • 4 -> 5 • 5 -> 6 -> 7 • 6 -> 7 • 7 -> 2 • 8 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 8 • 3 -> 4 -> 5 • 4 -> 5 • 5 -> 6 -> 7 • 6 -> 7 • 7 -> 2 • 8
3	iA2008	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 6 • 3 -> 4 -> 5 • 4 -> 3 • 5 -> 2 • 6 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 6 • 3 -> 4 -> 5 • 4 -> 3 • 5 -> 2 • 6
4	binA2008	<ul style="list-style-type: none"> • 1 -> 2 	<ul style="list-style-type: none"> • 1 -> 2

Tabel 2 Perbandingan *adjacency list* manual dan menggunakan aplikasi

No	Nama Program	Adjacency List Manual	Adjacency List Aplikasi
		<ul style="list-style-type: none"> • 2 -> 3 -> 9 • 3 -> 4 -> 5 • 4 -> 5 • 5 -> 6 -> 7 • 6 -> 8 • 7 -> 8 • 8 -> 2 • 9 -> 10 -> 11 • 10 -> 12 • 11 -> 12 • 12 	<ul style="list-style-type: none"> • 2 -> 3 -> 9 • 3 -> 4 -> 5 • 4 -> 5 • 5 -> 6 -> 7 • 6 -> 8 • 7 -> 8 • 8 -> 2 • 9 -> 10 -> 11 • 10 -> 12 • 11 -> 12 • 12
5	bubA2008	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 8 • 3 -> 4 -> 7 • 4 -> 5 -> 6 • 5 -> 6 • 6 -> 3 • 7 -> 2 • 8 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 8 • 3 -> 4 -> 7 • 4 -> 5 -> 6 • 5 -> 6 • 6 -> 3 • 7 -> 2 • 8
6	eB2002	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 4 -> 5 -> 6 -> 7 • 3 -> 11 • 4 -> 11 • 5 -> 11 • 6 -> 8 -> 11 • 8 -> 9 -> 10 • 9 -> 10 • 10 -> 6 • 11 • 7 -> 12 -> 13 • 12 -> 14 • 13 -> 14 • 14 -> 15 -> 20 • 15 -> 16 -> 18 • 16 -> 19 • 18 -> 19 -> 19 • 19 -> 18 • 19 -> 20 -> 21 • 20 -> 21 • 21 -> 14 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 4 -> 5 -> 6 -> 11 • 3 -> 10 • 4 -> 10 • 5 -> 10 • 6 -> 7 -> 10 • 7 -> 8 -> 9 • 8 -> 9 • 9 -> 6 • 10 • 11 -> 12 -> 13 • 12 -> 14 • 13 -> 14 • 14 -> 15 -> 20 • 15 -> 16 -> 17 • 16 -> 19 • 17 -> 18 -> 19 • 18 -> 17 • 19 -> 20 -> 21 • 20 -> 21 • 21 -> 14

Tabel 2 Perbandingan *adjacency list* manual dan menggunakan aplikasi

No	Nama Program	Adjacency List Manual	Adjacency List Aplikasi
7	gA2008	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 4 • 3 -> 9 • 4 -> 5 -> 9 • 5 -> 6 -> 7 • 6 -> 8 • 7 -> 8 • 8 -> 4 • 9 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 4 • 3 -> 9 • 4 -> 5 -> 9 • 5 -> 6 -> 7 • 6 -> 8 • 7 -> 8 • 8 -> 4 • 9
8	qB2002	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 10 • 3 -> 4 -> 10 • 4 -> 5 -> 6 • 5 -> 4 • 6 -> 7 -> 10 • 7 -> 8 -> 9 • 8 -> 9 • 9 -> 6 • 10 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 10 • 3 -> 4 -> 10 • 4 -> 5 -> 6 • 5 -> 4 • 6 -> 7 -> 10 • 7 -> 8 -> 9 • 8 -> 9 • 9 -> 6 • 10
9	fG2011	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 20 • 3 -> 4 -> 11 • 4 -> 5 -> 6 • 5 -> 7 • 6 -> 7 • 7 -> 8 -> 9 • 8 -> 10 • 9 -> 10 • 10 -> 12 -> 13 • 11 -> 10 • 12 -> 14 -> 15 • 14 -> 16 • 15 -> 16 • 16 -> 17 -> 18 • 13 -> 16 • 17 -> 19 • 18 -> 19 • 19 -> 2 • 20 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 20 • 3 -> 4 -> 11 • 4 -> 5 -> 6 • 5 -> 7 • 6 -> 7 • 7 -> 8 -> 9 • 8 -> 10 • 9 -> 10 • 10 -> 12 -> 16 • 11 -> 10 • 12 -> 13 -> 14 • 13 -> 15 • 14 -> 15 • 15 -> 17 -> 18 • 16 -> 15 • 17 -> 19 • 18 -> 19 • 19 -> 2 • 20
10	fmH2015	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 4 -> 5 	<ul style="list-style-type: none"> • 1 -> 2 • 2 -> 3 -> 4 -> 5

Tabel 2 Perbandingan *adjacency list* manual dan menggunakan aplikasi

No	Nama Program	Adjacency List Manual	Adjacency List Aplikasi
		<ul style="list-style-type: none"> • 3 -> 6 • 4 -> 6 • 5 -> 6 • 6 -> 7 -> 10 • 7 -> 8 -> 9 • 8 -> 10 • 9 -> 10 • 10 	<ul style="list-style-type: none"> • 3 -> 6 • 4 -> 6 • 5 -> 6 • 6 -> 7 -> 10 • 7 -> 8 -> 9 • 8 -> 10 • 9 -> 10 • 10

Uji efisiensi dilakukan dengan membandingkan waktu eksekusi yang dilakukan secara manual dengan waktu eksekusi oleh aplikasi. Pengukuran waktu eksekusi tidak bergantung pada jaringan karena aplikasi dijalankan di lokal komputer. Penguji secara manual terdiri dari dua orang yang berprofesi sebagai pengembang sistem. Hasil perbandingan waktu yang dibutuhkan untuk melakukan pembangkitan secara manual dan oleh aplikasi dapat dilihat pada

Tabel 3. Program uji nomor 1 tA2008 tidak ada waktu eksekusi secara manual karena program tersebut sudah digunakan sebagai contoh.

Waktu yang dibutuhkan aplikasi untuk membangkitkan CFG, melakukan instrumentasi, menghitung *cyclomatic complexity*, dan membangkitkan semua kemungkinan jalur rata-rata adalah 0.228 detik. Sedangkan jika hal tersebut dilakukan secara manual, akan menghabiskan waktu rata-rata 383.28 detik atau 6 menit 23 detik. Jumlah waktu yang dibutuhkan juga akan semakin meningkat ketika kode program semakin kompleks seperti pada program uji nomor 6 eB2002 yang dapat menghabiskan waktu rata-rata 659.43 detik atau 10 menit 59 detik. Sedangkan tidak berpengaruh ketika dieksekusi menggunakan aplikasi yang hanya menghabiskan waktu 0.22 detik. Dalam pengujian efisiensi, waktu yang diperoleh dari pengujian melalui aplikasi adalah waktu eksekusi dari aplikasi yang dijalankan di dalam komputer lokal tanpa menggunakan koneksi internet.

Tabel 3 Perbandingan waktu eksekusi secara manual dan menggunakan aplikasi

No	Nama Program	Waktu Eksekusi Aplikasi (detik)	Waktu Eksekusi Manual (detik)		
			Penguji 1	Penguji 2	Rata-Rata
1	tA2008	0.19	-	-	-
2	mmA2008	0.26	558.23	407.9	483.07
3	iA2008	0.11	234.55	358.51	296.53
4	binA2008	0.48	384.64	526.59	455.62
5	bubA2008	0.11	448.64	207.97	328.31
6	eB2002	0.22	686.59	632.27	659.43
7	gA2008	0.21	492.93	293.01	392.97

Tabel 3 Perbandingan waktu eksekusi secara manual dan menggunakan aplikasi

No	Nama Program	Waktu Eksekusi Aplikasi (detik)	Waktu Ekseseksi Manual (detik)		
			Penguji 1	Penguji 2	Rata-Rata
8	qB2002	0.16	222.25	280.86	251.56
9	fmH2015	0.2	201.87	223.07	212.47
10	fG2011	0.34	335.92	403.27	369.6
Rata-Rata		0.228	396.18	370.38	383.28

SIMPULAN DAN SARAN

Simpulan

Penelitian ini berhasil membangun sebuah aplikasi yang dapat digunakan untuk melakukan instrumentasi secara otomatis, membangkitkan CFG, menghitung *cyclomatic complexity*, dan membangkitkan segala kemungkinan jalur yang dapat dilewati dari kode program matlab untuk *path testing*..

Hasil penelitian menunjukkan bahwa kecepatan eksekusi jauh lebih cepat dibandingkan dilakukan secara manual sehingga dapat menghemat sumber daya dalam melakukan pengujian perangkat lunak.

Saran

Penelitian selanjutnya diharapkan dapat mengakomodir bahasa lain selain bahasa matlab. Selain itu, instrumentasi juga dapat dilakukan dinamis sesuai dengan kondisi yang dibutuhkan. Seperti yang dilakukan oleh Hermadi, selain menyimpan informasi jalur mana yang dilewati, instrumentasi yang dilakukan pada penelitian tersebut juga menyisipkan kode program untuk menghitung nilai *fitness*.

DAFTAR PUSTAKA

- Alimucaj A. 2009. Control Flow Graph Generator Documentation. [Internet]. [diunduh 2017 Juli 19]. Tersedia pada: <http://eclipsefcg.sourceforge.net/documentation.pdf>.
- Arkeman Y, Herdiyeni Y, Hermadi I, dan Laxmi G F. 2014. Algoritma Genetika Tujuan Jamak (MultiObjective Genetic Algorithm. Bogor (ID). IPB Press.
- Basu, A. 2015. *Software Quality Assurance, Testing and Metrics*. PHI Learning Privat Limited. [Internet]. [diunduh 2017 Agustus 14]. Tersedia pada: <https://github.com/JamieDixon/GraphViz-C-Sharp-Wrapper>.
- Dixon, J. 2013. "Graphviz.Net C# Wrapper". [Internet]. [diunduh 2017 Desember 22]. Tersedia pada: <https://github.com/JamieDixon/GraphViz-C-Sharp-Wrapper>.

- Ellson, J *et al.* 2003. "Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools". [Internet]. [diunduh 2017 Agustus 14]. Tersedia pada: <https://github.com/JamieDixon/GraphVizC-Sharp-Wrapper>.
- Gansner Emden R., Koutsofios, Eleftherios, dan North, Stephen. 2015. "Drawing graphs with dot". [Internet]. [diunduh 2017 Desember 25]. Tersedia pada: www.graphviz.org/pdf/dotguide.pdf.
- Hartwell, J. 2017. *C# and XML Primer*. Apress.
- Hermadi, I. 2015. "Path Testing using Genetic Algorithm". Disertasi. University of New South Wales.
- Houcq, David. 2015. "Intoruction To MATLAB For Engineering Students". [Internet]. [diunduh 2017 Desember 30]. Tersedia pada: <https://www.mccormick.northwestern.edu/documents/students/undergraduate/introductionto-matlab.pdf>.
- Kumar, D dan Mishra, K K. 2016. "The Impacts of Test Automation on Software's Cost, Quality and Time to Market" dalam: *Procedia Computer Science* 79,pp. 8–15. [Internet]. [diunduh 2017 Agustus 20]. Tersedia pada: <http://www.sciencedirect.com/science/article/pii/S1877050916001277>.
- Myers, G J, Sandler, C, dan Badgett, T. 2012. *The Art of Software Testing*. John Willey dan Sons, Inc, Hoboken, New York. [Internet]. [diunduh 2017 Agustus 14]. Tersedia pada: <https://books.google.co.id/books>.
- Suffos, S. 2015. "Matlab Parser". [Internet]. [diunduh 2017 Desember 22]. Tersedia pada: <https://github.com/samuel-suffos/matlabparser>.
- Tikir, M M dan Hollingsworth, J K. 2011. "Efficient Instrumentation for Code Coverage Testing" dalam: *International Journal of Software Engineering and Its Applications*. [Internet]. [diunduh 2017 Agustus 21]. Tersedia pada: https://www.researchgate.net/publication/2835608_Efficient_Instrumentation_for_Code_Coverage_Testing.
- Watson, A H dan McCabe, T J. 1996. "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric)" dalam: *NIST Special Publication*. [Internet]. [diunduh 2017 Agustus 14]. Tersedia pada: <http://www.mccabe.com/pdf/mccabe-nist235r.pdf>.

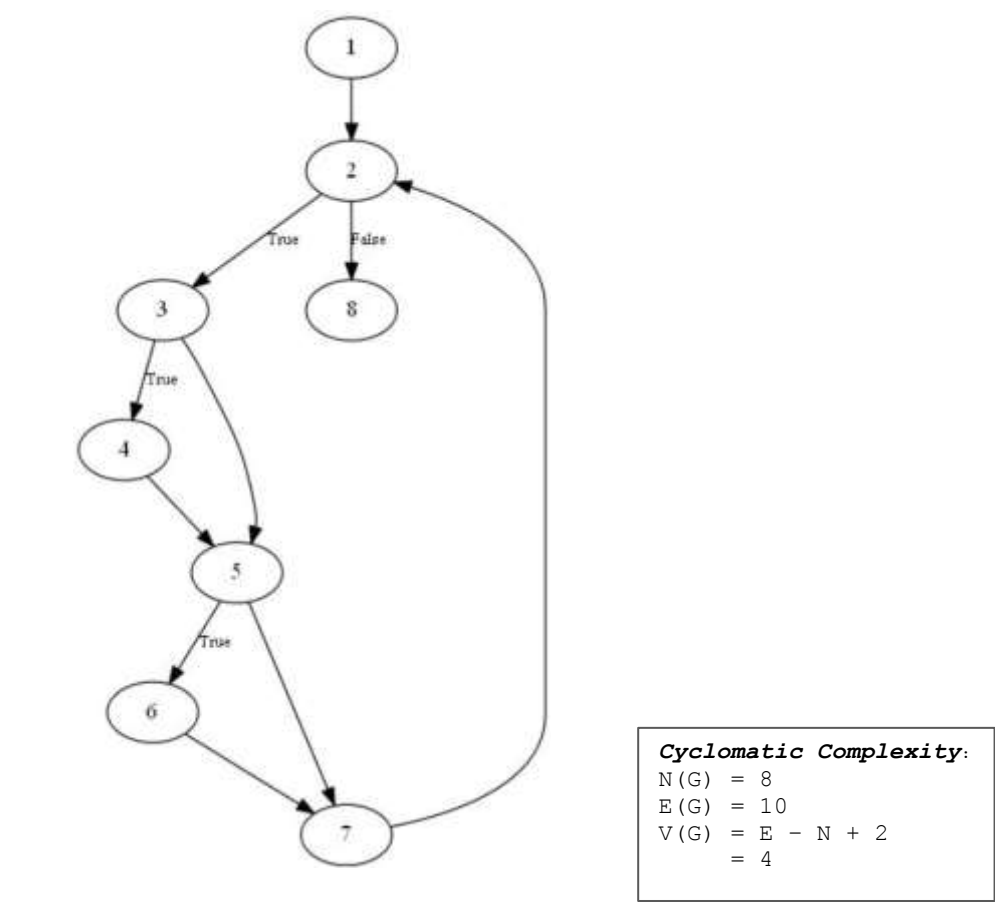
Lampiran 1 Kode program mmA2008

1	Node 1	function miniMaxi = minimaxi(num)
2		numLength = length(num);
3		mini = num(1);
4		maxi = num(1);
5		idx = 2;
6	Node 2	while (idx <= numLength) % Branching #1
7	Node 3	if maxi < num(idx) % Branching #2
8	Node 4	maxi = num(idx);
9		end
10	Node 5	if mini > num(idx) % Branching #3
11	Node 6	mini = num(idx);
12		end
13	Node 7	idx = idx+1;
14		end
15		miniMaxi = [mini maxi];
16	Node 8	end

Lampiran 2 Hasil instrumentasi mmA2008

1	Node 1	function [traversedPath,miniMaxi] = minimaxi(num)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		numLength = length(num);
5		mini = num(1);
6		maxi = num(1);
7		idx = 2;
8		% instrument Branch # 1
9	Node 2	traversedPath = [traversedPath '2 '];
10		while (idx <= numLength) % Branching #1
11		traversedPath = [traversedPath '(T) '];
12		% instrument Branch # 2
13	Node 3	traversedPath = [traversedPath '3 '];
14	Node 4	if maxi < num(idx) % Branching #2
15		traversedPath = [traversedPath '(T) '];
16		traversedPath = [traversedPath '4 '];
17		maxi = num(idx);
18		end
19		% instrument Branch # 3
20	Node 5	traversedPath = [traversedPath '5 '];
21	Node 6	if mini > num(idx) % Branching #3
22		traversedPath = [traversedPath '(T) '];
23		traversedPath = [traversedPath '6 '];
24		mini = num(idx);
25		end
26	Node 7	traversedPath = [traversedPath '7 '];
27		idx = idx+1;
28		traversedPath = [traversedPath '2 '];
29		end
30		miniMaxi = [mini maxi];
31		traversedPath = [traversedPath '(F) '];
32	Node 8	traversedPath = [traversedPath '8 '];
33		end

Lampiran 3 CFG dan *Cyclomatic Complexity* mmA2008



Lampiran 4 Semua kemungkinan jalur mmA2008

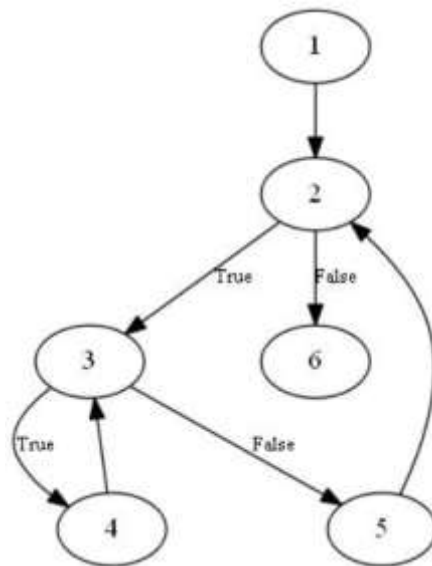
- 1 2 (T) 3 (T) 4 5 (T) 6 7 2 (F) 8
- 1 2 (T) 3 (T) 4 5 (F) 7 2 (F) 8
- 1 2 (T) 3 (F) 5 (T) 6 7 2 (F) 8
- 1 2 (T) 3 (F) 5 (F) 7 2 (F) 8
- 1 2 (F) 8

Lampiran 5 Kode program iA2008

1	Node 1	function sortedArray = insertion(anyArray)
2		k = 1; % The smallest integer increment
3		n = length(anyArray);
4		i = 2;
5	Node 2	for i=2:n
6		x = anyArray(i);
7		j = i + 1;
8	Node 3	while ((j > 0) & (anyArray(j) > x)),
9	Node 4	anyArray(j+1) = anyArray(j);
10		j = j - 1;
11		end
12	Node 5	anyArray(j+1) = x;
13		end
14		sortedArray = anyArray;
15	Node 6	end

Lampiran 6 Hasil instrumentasi iA2008

1	Node 1	function [traversedPath,sortedArray] = insertion(anyArray)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		k = 1; % The smallest integer increment
5		n = length(anyArray);
6		i = 2;
7		% instrument Branch # 1
8		traversedPath = [traversedPath '2 '];
9	Node 2	for i=2:n
10		x = anyArray(i);
11		j = i + 1;
12		traversedPath = [traversedPath '(T) '];
13		% instrument Branch # 2
14		traversedPath = [traversedPath '3 '];
15	Node 3	while ((j > 0) & (anyArray(j) > x)),
16		traversedPath = [traversedPath '(T) '];
17		traversedPath = [traversedPath '4 '];
18	Node 4	anyArray(j+1) = anyArray(j);
19		j = j - 1;
20		traversedPath = [traversedPath '3 '];
21		end
22		traversedPath = [traversedPath '(F) '];
23		traversedPath = [traversedPath '5 '];
24	Node 5	anyArray(j+1) = x;
25		traversedPath = [traversedPath '2 '];
26		end
27		sortedArray = anyArray;
28		traversedPath = [traversedPath '(F) '];
29		traversedPath = [traversedPath '6 '];
30	Node 6	end

Lampiran 7 CFG dan *Cyclomatic Complexity* iA2008**Cyclomatic Complexity:**

$$\begin{aligned}
 N(G) &= 6 \\
 E(G) &= 7 \\
 V(G) &= E - N + 2 \\
 &= 3
 \end{aligned}$$

Lampiran 8 Semua kemungkinan jalur iA2008

- 1 2 (T) 3 (T) 4 3 (F) 5 2 (F) 6
- 1 2 (T) 3 (F) 5 2 (F) 6
- 1 2 (F) 6

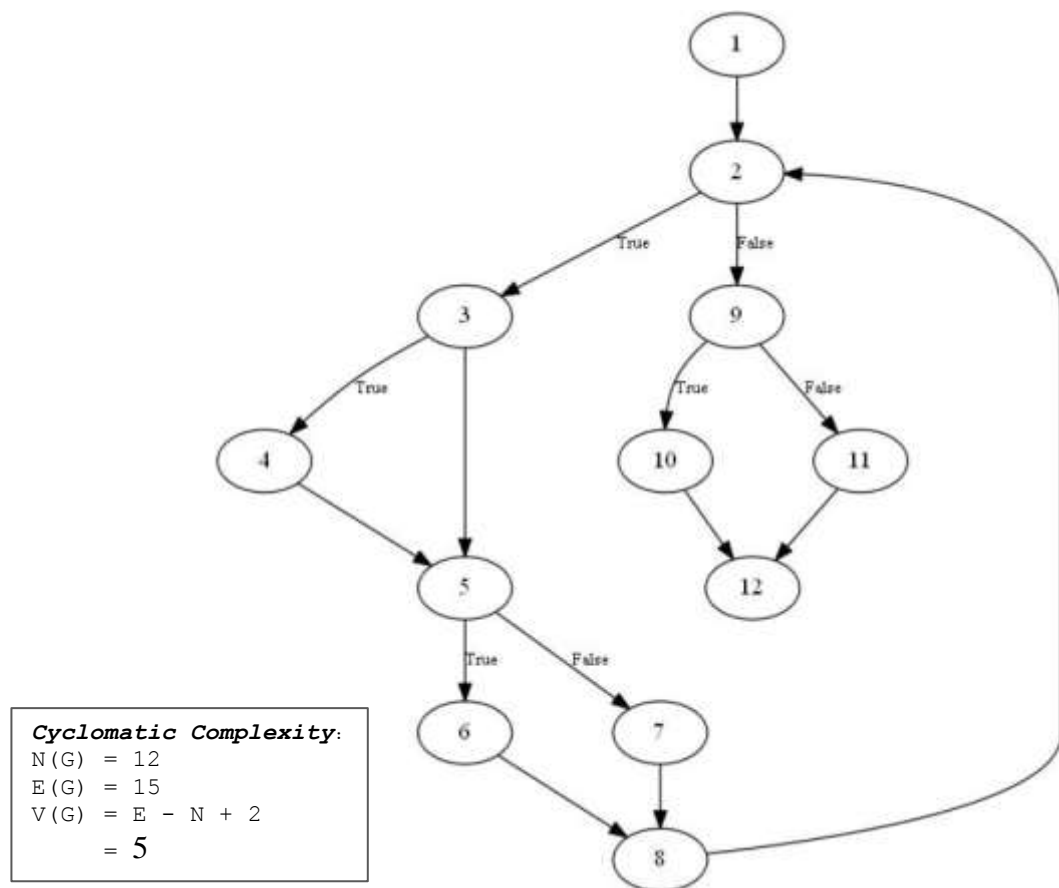
Lampiran 9 Kode program binA2008

1	Node 1	function itemIndex = binary(itemNumbers)
2		item = itemNumbers(1);
3		numbers = itemNumbers(1,2:end);
4		lowerIdx = 1;
5		upperIdx = length(numbers);
6	Node 2	while (lowerIdx ~= upperIdx), % Branch # 1
7		temp = lowerIdx + upperIdx; % additional statement
8	Node 3	if (mod(temp, 2) ~= 0),
9	Node 4	temp = temp - 1;
10		end % additional statement
11		idx = temp / 2;
12	Node 5	if (numbers(idx) < item), % Branch # 2
13	Node 6	lowerIdx = idx + 1;
14		else
15	Node 7	upperIdx = idx;
16	Node 8	end
17		end
18		% Additional code that returns -1 if the item is not found
19	Node 9	if (item == numbers(lowerIdx)),
20	Node 10	temIndex = lowerIdx;
21		else
22	Node 11	itemIndex = -1;
23		end
24	Node 12	end

Lampiran 10 Hasil instrumentasi binA2008

1	Node 1	function [traversedPath,itemIndex] = binary(itemNumbers)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		item = itemNumbers(1);
5		numbers = itemNumbers(1,2:end);
6		lowerIdx = 1;
7		upperIdx = length(numbers);
8		% instrument Branch # 1
9		traversedPath = [traversedPath '2 '];
10	Node 2	while (lowerIdx ~= upperIdx), % Branch # 1
11		temp = lowerIdx + upperIdx; % additional statement
12		traversedPath = [traversedPath '(T) '];
13		% instrument Branch # 2
14		traversedPath = [traversedPath '3 '];
15	Node 3	if (mod(temp, 2) ~= 0),
16		traversedPath = [traversedPath '(T) '];
17		traversedPath = [traversedPath '4 '];
18	Node 4	temp = temp - 1;
19		end % additional statement
20		idx = temp / 2;
21		% instrument Branch # 3
22		traversedPath = [traversedPath '5 '];
23	Node 5	if (numbers(idx) < item), % Branch # 2
24		traversedPath = [traversedPath '(T) '];
25		traversedPath = [traversedPath '6 '];
26	Node 6	lowerIdx = idx + 1;
27		else
28		traversedPath = [traversedPath '(F) '];
29		traversedPath = [traversedPath '7 '];
30	Node 7	upperIdx = idx;
31	Node 8	end
32		traversedPath = [traversedPath '8 '];
33		traversedPath = [traversedPath '2 '];
34		end
35		% Additional code that returns -1 if the item is not found
36		traversedPath = [traversedPath '(F) '];
37		% instrument Branch # 4
38		traversedPath = [traversedPath '9 '];

39	Node 9	if (item == numbers(lowerIdx)),
40		traversedPath = [traversedPath '(T) '];
41		traversedPath = [traversedPath '10 '];
42	Node 10	temIndex = lowerIdx;
43		else
44		traversedPath = [traversedPath '(F) '];
45		traversedPath = [traversedPath '11 '];
46	Node 11	itemIndex = -1;
47		end
48		traversedPath = [traversedPath '12 '];
49	Node 12	end

Lampiran 11 CFG dan *Cyclomatic Complexity* binA2008

Lampiran 12 Semua kemungkinan jalur binA2008

- 1 2 (T) 3 (T) 4 5 (T) 6 8 2 (F) 9 (T) 10 12
- 1 2 (T) 3 (T) 4 5 (T) 6 8 2 (F) 9 (F) 11 12
- 1 2 (T) 3 (T) 4 5 (F) 7 8 2 (F) 9 (T) 10 12
- 1 2 (T) 3 (T) 4 5 (F) 7 8 2 (F) 9 (F) 11 12
- 1 2 (T) 3 5 (T) 6 8 2 (F) 9 (T) 10 12
- 1 2 (T) 3 5 (T) 6 8 2 (F) 9 (F) 11 12
- 1 2 (T) 3 5 (F) 7 8 2 (F) 9 (T) 10 12
- 1 2 (T) 3 5 (F) 7 8 2 (F) 9 (F) 11 12
- 1 2 (F) 9 (T) 10 12
- 1 2 (F) 9 (F) 11 12

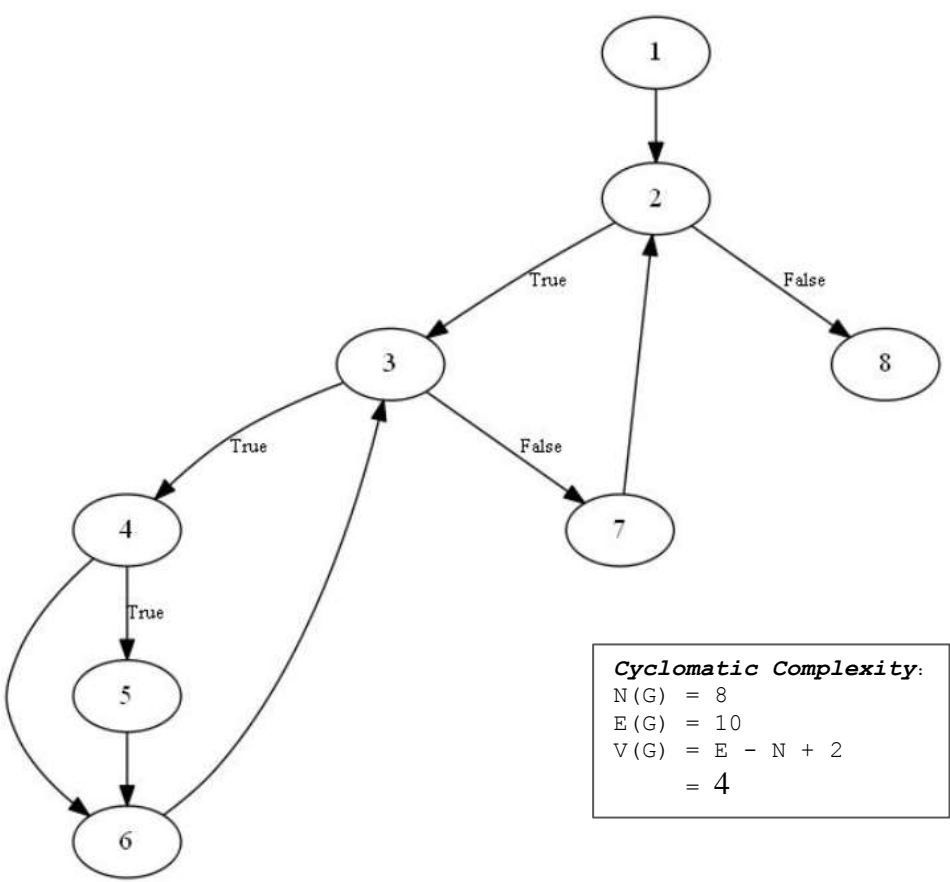
Lampiran 13 Kode program bubA2008

1	Node 1	function sortedArray = bubble(anyArray)
2		sorted = 0; % 0 means false
3		i = 1; n = length(anyArray);
4	Node 2	while ((i <= (n-1)) && ~sorted), % Branch # 1
5		sorted = 1;
6		j = n;
7	Node 3	for j=n:-1:i+1 % Branch # 2
8	Node 4	if (anyArray(j) < anyArray(j-1)) % Branch # 3
9		%exchange(anyArray(j), anyArray(j-1));
10	Node 5	temp = anyArray(j);
11		anyArray(j) = anyArray(j-1);
12		anyArray(j-1) = temp;
13		sorted = 0;
14	Node 6	end
15	Node 7	end
16		i = i + 1;
17		end
18		sortedArray = anyArray;
19	Node 8	end

Lampiran 14 Hasil instrumentasi bubA2008

1	Node 1	function [traversedPath,sortedArray] = bubble(anyArray)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		sorted = 0; % 0 means false
5		i = 1; n = length(anyArray);
6		% instrument Branch # 1
7		traversedPath = [traversedPath '2 '];
8	Node 2	while ((i <= (n-1)) && ~sorted), % Branch # 1
9		sorted = 1;
10		j = n;
11		traversedPath = [traversedPath '(T) '];
12		% instrument Branch # 2
13		traversedPath = [traversedPath '3 '];
14	Node 3	for j=n:-1:i+1 % Branch # 2
15		traversedPath = [traversedPath '(T) '];
16		% instrument Branch # 3
17		traversedPath = [traversedPath '4 '];
18	Node 4	if (anyArray(j) < anyArray(j-1)) % Branch # 3
19		%exchange(anyArray(j), anyArray(j-1));
20		traversedPath = [traversedPath '(T) '];
21		traversedPath = [traversedPath '5 '];
22	Node 5	temp = anyArray(j);
23		anyArray(j) = anyArray(j-1);
24		anyArray(j-1) = temp;
25		sorted = 0;
26	Node 6	end
27		traversedPath = [traversedPath '6 '];
28		traversedPath = [traversedPath '3 '];
29	Node 7	end
30		traversedPath = [traversedPath '(F) '];
31		traversedPath = [traversedPath '7 '];
32		i = i + 1;
33		traversedPath = [traversedPath '2 '];
34		end
35		sortedArray = anyArray;
36		traversedPath = [traversedPath '(F) '];
37		traversedPath = [traversedPath '8 '];
38	Node 8	end

Lampiran 15 CFG dan *Cyclomatic Complexity* bubA2008



Lampiran 16 Semua kemungkinan jalur bubA2008

- 1 2 (T) 3 (T) 4 (T) 5 6 3 (F) 7 2 (F) 8
- 1 2 (T) 3 (T) 4 6 3 (F) 7 2 (F) 8
- 1 2 (T) 3 (F) 7 2 (F) 8
- 1 2 (F) 8

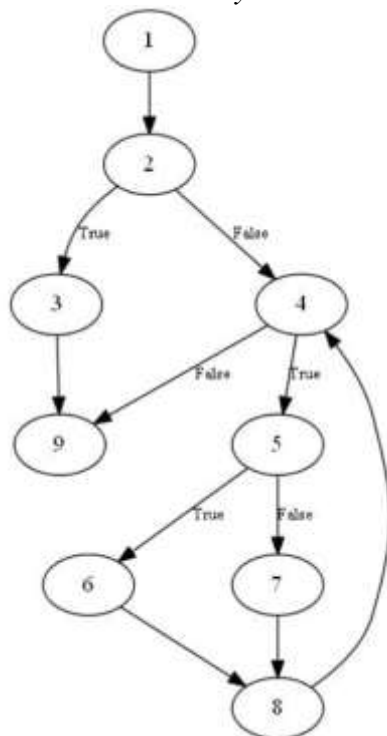
Lampiran 17 Kode program gA2008

1	Node 1	function y = gcd(number)
2		a = number(1);
3		b = number(2);
4	Node 2	if (a == 0),
5	Node 3	y = b;
6		else
7	Node 4	while b ~= 0
8	Node 5	if a > b
9	Node 6	a = a - b;
10		else
11	Node 7	b = b - a;
12	Node 8	end
13		end
14		y = a;
15		end
16	Node 9	end

Lampiran 18 Hasil instrumentasi gA2008

1	Node 1	function [traversedPath,y] = gcd(number)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		a = number(1);
5		b = number(2);
6		% instrument Branch # 1
7		traversedPath = [traversedPath '2 '];
8	Node 2	if (a == 0),
9		traversedPath = [traversedPath '(T) '];
10		traversedPath = [traversedPath '3 '];
11	Node 3	y = b;
12		else
13		traversedPath = [traversedPath '(F) '];
14		% instrument Branch # 2
15		traversedPath = [traversedPath '4 '];
16	Node 4	while b ~= 0
17		traversedPath = [traversedPath '(T) '];
18		% instrument Branch # 3
19		traversedPath = [traversedPath '5 '];
20	Node 5	if a > b
21		traversedPath = [traversedPath '(T) '];
22		traversedPath = [traversedPath '6 '];
23	Node 6	a = a - b;
24		else
25		traversedPath = [traversedPath '(F) '];
26		traversedPath = [traversedPath '7 '];
27	Node 7	b = b - a;
28	Node 8	end
29		traversedPath = [traversedPath '8 '];
30		traversedPath = [traversedPath '4 '];
31		end
32		y = a;
33		end
34		traversedPath = [traversedPath '(F) '];
35		traversedPath = [traversedPath '9 '];
36	Node 9	end

Lampiran 19 CFG dan Cyclomatic Complexity gA2008

**Cyclomatic Complexity:**

$$\begin{aligned}
 N(G) &= 9 \\
 E(G) &= 11 \\
 V(G) &= E - N + 2 \\
 &= 4
 \end{aligned}$$

Lampiran 20 Semua kemungkinan jalur gA2008

- 1 2 (T) 3 9
- 1 2 (F) 4 (T) 5 (T) 6 8 4 (F) 9
- 1 2 (F) 4 (T) 5 (F) 7 8 4 (F) 9
- 1 2 (F) 4 (F) 9

Lampiran 21 Kode program eB2002

1	Node 1	function result = expintBueno2002(numbersIn)
2		n = numbersIn(1); % integer
3		x = numbersIn(2); % floa
4		MAXIT = 100;
5		EULER = 0.5772156649;
6		FPMIN = 1.0e-30;
7		EPS = 1.0e-7;
8		nm1 = n - 1;
9	Node 2	if (n < 0 x < 0.0 (x == 0.0 && (n == 0.0 n==1)))
10	Node 3	result = 0;
11		% disp('bad arguments in expintBueno2002');
12		elseif (n == 0)
13	Node 4	result = exp(-x)/x;
14		elseif (x == 0.0)
15	Node 5	result = 1.0/nm1; % strangy: what is nm1?
16		elseif (x > 1.0)
17		b = x + n;
18		c = 1.0 / FPMIN;
19		d = 1.0 / b;
20		h = d;
21	Node 6	for i=1 : MAXIT
22		a = -i * (nm1 + i);
23		b = b + 2.0;
24		d = 1.0 / (a*d+b);
25		c = b + a / c;
26		del = c * d;
27		h = h * del;
28	Node 7	if (abs(del-1.0) < EPS) % abs is fabs in C
29	Node 8	result = h * exp(-x);
30		return;
31	Node 9	end
32		end
33		disp('continued fraction failed in expint');
34		else
35	Node 10	% ans = (nm1!=0 ? 1.0/nm1 : -log(x)-EULER);
36		% is interpreted as follows
37	Node 11	if (nm1 ~= 0)
38	Node 12	result = 1.0 / nm1;
39		else
40	Node 13	result = -log(x)-EULER;
41		end
42		fact = 1.0;
43	Node 14	for i = 1 : MAXIT
44		fact = fact * (-x / i);
45	Node 15	if (i ~= nm1)
46	Node 16	del = -fact / (i - nm1);
47		else
48		psi = -EULER;
49	Node 17	for ii = 1 : nm1
50	Node 18	psi = psi + (1/ii);
51		end
52		del = fact * (-log(x) + psi);
53		end
54		result = result + del;
55	Node 19	if (abs(del) < abs(result) * EPS) % abs is fabs in C
56	Node 20	return;
57	Node 21	end

58		end
59		disp('series failed in expint');
60		end
61	Node 10	end

Lampiran 22 Hasil instrumentasi eB2002

1	Node 1	function [traversedPath,result] = expintBueno2002(numbersIn)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		n = numbersIn(1); % integer
5		x = numbersIn(2); % floa
6		MAXIT = 100;
7		EULER = 0.5772156649;
8		FPMIN = 1.0e-30;
9		EPS = 1.0e-7;
10		nm1 = n - 1;
11		% instrument Branch # 1
12		traversedPath = [traversedPath '2 '];
13	Node 2	if (n < 0 x < 0.0 (x == 0.0 && (n == 0.0 n==1)))
14		traversedPath = [traversedPath '(T) '];
15		traversedPath = [traversedPath '3 '];
16	Node 3	result = 0;
17		% disp('bad arguments in expintBueno2002');
18		elseif (n == 0)
19		traversedPath = [traversedPath '(T) '];
20		traversedPath = [traversedPath '4 '];
21	Node 4	result = exp(-x)/x;
22		elseif (x == 0.0)
23		traversedPath = [traversedPath '(T) '];
24		traversedPath = [traversedPath '5 '];
25	Node 5	result = 1.0/nm1; % strangy: what is nm1?
26		elseif (x > 1.0)
27		b = x + n;
28		c = 1.0 / FPMIN;
29		d = 1.0 / b;
30		h = d;
31		traversedPath = [traversedPath '(T) '];
32		% instrument Branch # 2
33		traversedPath = [traversedPath '6 '];
34	Node 6	for i=1 : MAXIT
35		a = -i * (nm1 + i);
36		b = b + 2.0;
37		d = 1.0 / (a*d+b);
38		c = b + a / c;
39		del = c * d;
40		h = h * del;
41		traversedPath = [traversedPath '(T) '];
42		% instrument Branch # 3
43		traversedPath = [traversedPath '7 '];
44	Node 7	if (abs(del-1.0) < EPS) % abs is fabs in C
45		traversedPath = [traversedPath '(T) '];
46		traversedPath = [traversedPath '8 '];
47	Node 8	result = h * exp(-x);
48		return;
49	Node 9	end
50		traversedPath = [traversedPath '9 '];
51		traversedPath = [traversedPath '6 '];
52		traversedPath = [traversedPath '14 '];
53		end
54		disp('continued fraction failed in expint');
55		else
56		% ans = (nm1!=0 ? 1.0/nm1 : -log(x)-EULER);
57		% is interpreted as follows
58		traversedPath = [traversedPath '(F) '];
59		% instrument Branch # 4
60		traversedPath = [traversedPath '11 '];
61	Node 11	if (nm1 ~= 0)
62		traversedPath = [traversedPath '(T) '];

63		traversedPath = [traversedPath '12 '];
64	Node 12	result = 1.0 / nml;
65		else
66		traversedPath = [traversedPath '(F) '];
67		traversedPath = [traversedPath '13 '];
68	Node 13	result = -log(x)-EULER;
69		end
70		fact = 1.0;
71		% instrument Branch # 5
72		traversedPath = [traversedPath '14 '];
73	Node 14	for i = 1 : MAXIT
74		fact = fact * (-x / i);
75		traversedPath = [traversedPath '(T) '];
76		% instrument Branch # 6
77		traversedPath = [traversedPath '15 '];
78	Node 15	if (i ~= nml)
79		traversedPath = [traversedPath '(T) '];
80		traversedPath = [traversedPath '16 '];
81	Node 16	del = -fact / (i - nml);
82		else
83		psi = -EULER;
84		traversedPath = [traversedPath '(F) '];
85		% instrument Branch # 7
86		traversedPath = [traversedPath '17 '];
87	Node 17	for ii = 1 : nml
88		traversedPath = [traversedPath '(T) '];
89		traversedPath = [traversedPath '18 '];
90	Node 18	psi = psi + (1/ii);
91		traversedPath = [traversedPath '17 '];
92		end
93		del = fact * (-log(x) + psi);
94		end
95		result = result + del;
96		traversedPath = [traversedPath '(F) '];
97		% instrument Branch # 8
98		traversedPath = [traversedPath '19 '];
99	Node 19	if (abs(del) < abs(result) * EPS) % abs is fabs in C
100		traversedPath = [traversedPath '(T) '];
101		traversedPath = [traversedPath '20 '];
102	Node 20	return;
103	Node 21	end
104		traversedPath = [traversedPath '21 '];
105		end
106		disp('series failed in expint');
107		end
108		traversedPath = [traversedPath '(F) '];
109		traversedPath = [traversedPath '10 '];
120	Node 10	end

Lampiran 23 CFG dan *Cyclomatic Complexity* eB2002

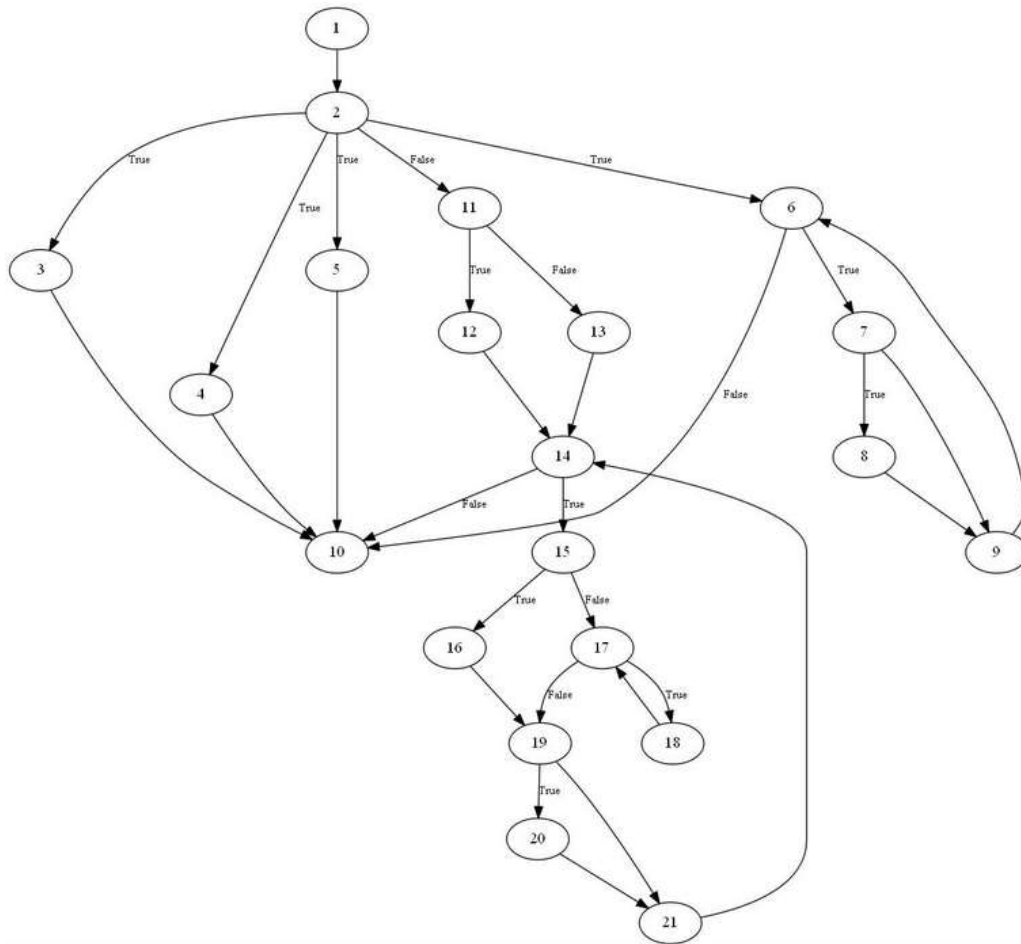
Cyclomatic Complexity:

$$N(G) = 21$$

$$E(G) = 31$$

$$V(G) = E - N + 2$$

$$= 12$$



Lampiran 24 Semua kemungkinan jalur eB2002

- 1 2 (T) 3 10
- 1 2 (T) 4 10
- 1 2 (T) 5 10
- 1 2 (T) 6 (T) 7 (T) 8 9 6 (F) 10
- 1 2 (T) 6 (T) 7 9 6 (F) 10
- 1 2 (T) 6 (F) 10
- 1 2 (F) 11 (T) 12 14 (T) 15 (T) 16 19 (T) 20 21 14 (F) 10
- 1 2 (F) 11 (T) 12 14 (T) 15 (T) 16 19 21 14 (F) 10
- 1 2 (F) 11 (T) 12 14 (T) 15 (F) 17 (T) 18 17 (F) 19 (T) 20 21 14 (F) 10
- 1 2 (F) 11 (T) 12 14 (T) 15 (F) 17 (T) 18 17 (F) 19 21 14 (F) 10
- 1 2 (F) 11 (T) 12 14 (T) 15 (F) 17 (F) 19 (T) 20 21 14 (F) 10
- 1 2 (F) 11 (T) 12 14 (T) 15 (F) 17 (F) 19 21 14 (F) 10
- 1 2 (F) 11 (T) 12 14 (F) 10
- 1 2 (F) 11 (F) 13 14 (T) 15 (T) 16 19 (T) 20 21 14 (F) 10
- 1 2 (F) 11 (F) 13 14 (T) 15 (T) 16 19 21 14 (F) 10
- 1 2 (F) 11 (F) 13 14 (T) 15 (F) 17 (T) 18 17 (F) 19 (T) 20 21 14 (F) 10
- 1 2 (F) 11 (F) 13 14 (T) 15 (F) 17 (T) 18 17 (F) 19 21 14 (F) 10
- 1 2 (F) 11 (F) 13 14 (T) 15 (F) 17 (F) 19 (T) 20 21 14 (F) 10
- 1 2 (F) 11 (F) 13 14 (T) 15 (F) 17 (F) 19 21 14 (F) 10
- 1 2 (F) 11 (F) 13 14 (F) 10

Lampiran 25 Kode program qB2002

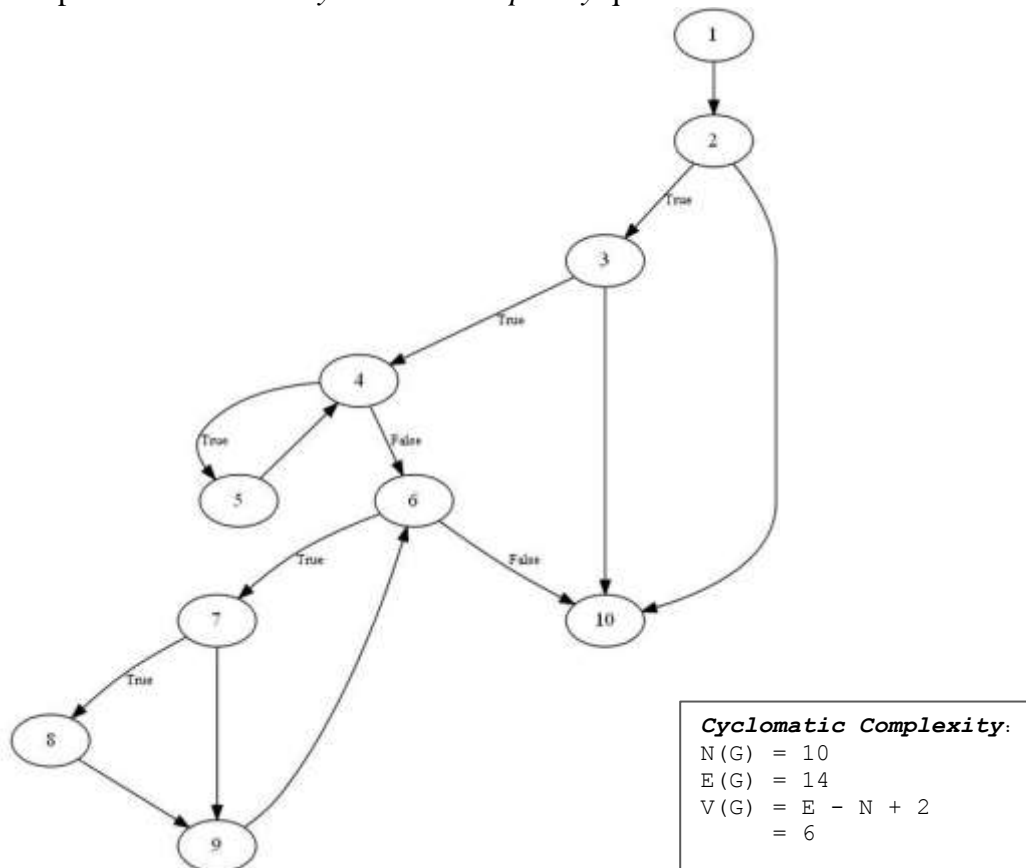
1	Node 1	function [q, r] = quotientBueno2002(operands)
2		n = operands(1); % First number
3		d = operands(2); % Second number
4		q = 0;
5	Node 2	if (d ~= 0)
6	Node 3	if ((d > 0) && (n > 0))
7		q = 0;
8		r = n;
9		t = d;
10	Node 4	while (r >= t)
11	Node 5	t = t * 2;
12		end
13	Node 6	while (t ~= d)
14	Node 7	q = q * 2;
15		t = t / 2;
16	Node 8	if (t <= r)
17	Node 9	r = r - t;
18		q = q + 1;
19		end
20		end
21		end
22		end
23	Node 10	end

Lampiran 26 Hasil instrumentasi qB2002

1	Node 1	function [traversedPath,q, r] = quotientBueno2002(operands)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		n = operands(1); % First number
5		d = operands(2); % Second number
6		q = 0;
7		% instrument Branch # 1
8		traversedPath = [traversedPath '2 '];
9	Node 2	if (d ~= 0)
10		traversedPath = [traversedPath '(T) '];
11		% instrument Branch # 2
12		traversedPath = [traversedPath '3 '];
13	Node 3	if ((d > 0) && (n > 0))
14		q = 0;
15		r = n;
16		t = d;
17		traversedPath = [traversedPath '(T) '];
18		% instrument Branch # 3
19		traversedPath = [traversedPath '4 '];
20	Node 4	while (r >= t)
21		traversedPath = [traversedPath '(T) '];
22		traversedPath = [traversedPath '5 '];
23	Node 5	t = t * 2;
24		traversedPath = [traversedPath '4 '];
25		end
26		traversedPath = [traversedPath '(F) '];
27		% instrument Branch # 4
28		traversedPath = [traversedPath '6 '];
29	Node 6	while (t ~= d)
30		q = q * 2;
31		t = t / 2;
32		traversedPath = [traversedPath '(T) '];
33		% instrument Branch # 5
34		traversedPath = [traversedPath '7 '];
35	Node 7	if (t <= r)
36		traversedPath = [traversedPath '(T) '];
37		traversedPath = [traversedPath '8 '];
38	Node 8	r = r - t;
39		q = q + 1;
40	Node 9	end

41		<code>traversedPath = [traversedPath '9 '];</code>
42		<code>traversedPath = [traversedPath '6 '];</code>
43		<code>end</code>
44		<code>end</code>
45		<code>end</code>
46		<code>traversedPath = [traversedPath '(F) '];</code>
47		<code>traversedPath = [traversedPath '10 '];</code>
48	Node 10	<code>end</code>

Lampiran 27 CFG dan *Cyclomatic Complexity* qB2002



Lampiran 28 Semua kemungkinan jalur qB2002

- 1 2 (T) 3 (T) 4 (T) 5 4 (F) 6 (T) 7 (T) 8 9 6 (F) 10
- 1 2 (T) 3 (T) 4 (T) 5 4 (F) 6 (T) 7 9 6 (F) 10
- 1 2 (T) 3 (T) 4 (T) 5 4 (F) 6 (F) 10
- 1 2 (T) 3 (T) 4 (F) 6 (T) 7 (T) 8 9 6 (F) 10
- 1 2 (T) 3 (T) 4 (F) 6 (T) 7 9 6 (F) 10
- 1 2 (T) 3 (T) 4 (F) 6 (F) 10
- 1 2 (T) 3 10
- 1 2 10

Lampiran 29 Kode program fG2011

1	Node 1	function pop = flexGong2011(depop)
2		[px,py]=size(depop);
3	Node 2	for i=1:px % Branch # 1
4		q=1;
5		p=depop(i,:);
6		lex_compat=p(1);
7		C_plus_plus=p(2);
8		fulltbl=p(3);
9		csize =p(4);
10		unspecified=p(5);
11		fullspd=p(6);
12		C_plus=p(7);
13		d(1,1)=2;
14		d(1,1)=1-1.001^(-d(1,1));
15		d(1,2)=lex_compat;
16		d(1,2)=1-1.001^(-d(1,2));
17		d(1,3)=0;
18		d(2,1)=2;
19		d(2,1)=1-1.001^(-d(2,1));
20		d(2,2)=C_plus_plus;
21		d(2,2)=1-1.001^(-d(2,2));
22		d(2,3)=0;
23		d(3,1)=2;
24		d(3,1)=1-1.001^(-d(3,1));
25		d(3,2)=fulltbl;
26		d(3,2)=1-1.001^(-d(3,2));
27		d(3,3)=0;
28		d(4,1)=abs(csize-unspecified)+2;
29		d(4,1)=1-1.001^(-d(4,1));
30		d(4,2)=2;
31		d(4,2)=1-1.001^(-d(4,2));
32		d(4,3)=0;
33		d(5,1)=2;
34		d(5,1)=1-1.001^(-d(5,1));
35		d(5,2)=fullspd;
36		d(5,2)=1-1.001^(-d(5,2));
37		d(5,3)=0;
38		d(6,1)=2;
39		d(6,1)=1-1.001^(-d(5,1));
40		d(6,2)=C_plus;
41		d(6,2)=1-1.001^(-d(5,2));
42		d(5,3)=0;
43		u=3*ones(1,6);
44	Node 3	if (lex_compat ~= 0) % Branch # 2
45		d(1,1)=0;
46		u(1)=1;
47	Node 4	if (C_plus_plus ~= 0) % Branch # 3
48	Node 5	flexerror = 'Can not use -+ with -l option';
49		d(2,1)=0;
50		else
51	Node 6	d(2,2)=0;
52		end
53	Node 7	if (fulltbl ~= 0) % Branch # 4
54	Node 8	flexerror='Can not use -f or -F with -l option';
55		d(3,1)=0;
56		else
57	Node 9	d(3,2)=0;
58		end
59		else
60	Node 11	d(1,2)=0;
61		end
62	Node 10	if (csize == unspecified) % Branch # 5
63		d(4,1)=0;
64	Node 12	if (fullspd ~= 0) % Branch # 6
65	Node 13	csize = 'DEFAULT_CSIZIE';
66		d(5,1)=0;
67		else
68	Node 14	d(5,2)=0;

69		csize = csize;
70		end
71		else
72	Node 16	d(4,2)=0;
73		end
74	Node 15	if (C_plus ~= 0) % Branch # 7
75	Node 17	suffix='cc';
76		d(6,1)=0;
77		else
78	Node 18	d(6,2)=0;
79		suffix='c';
80		outfilename = 'outfile_path';
81	Node 19	end
82		pop(i,:) = p;
83		end
84	Node 20	end

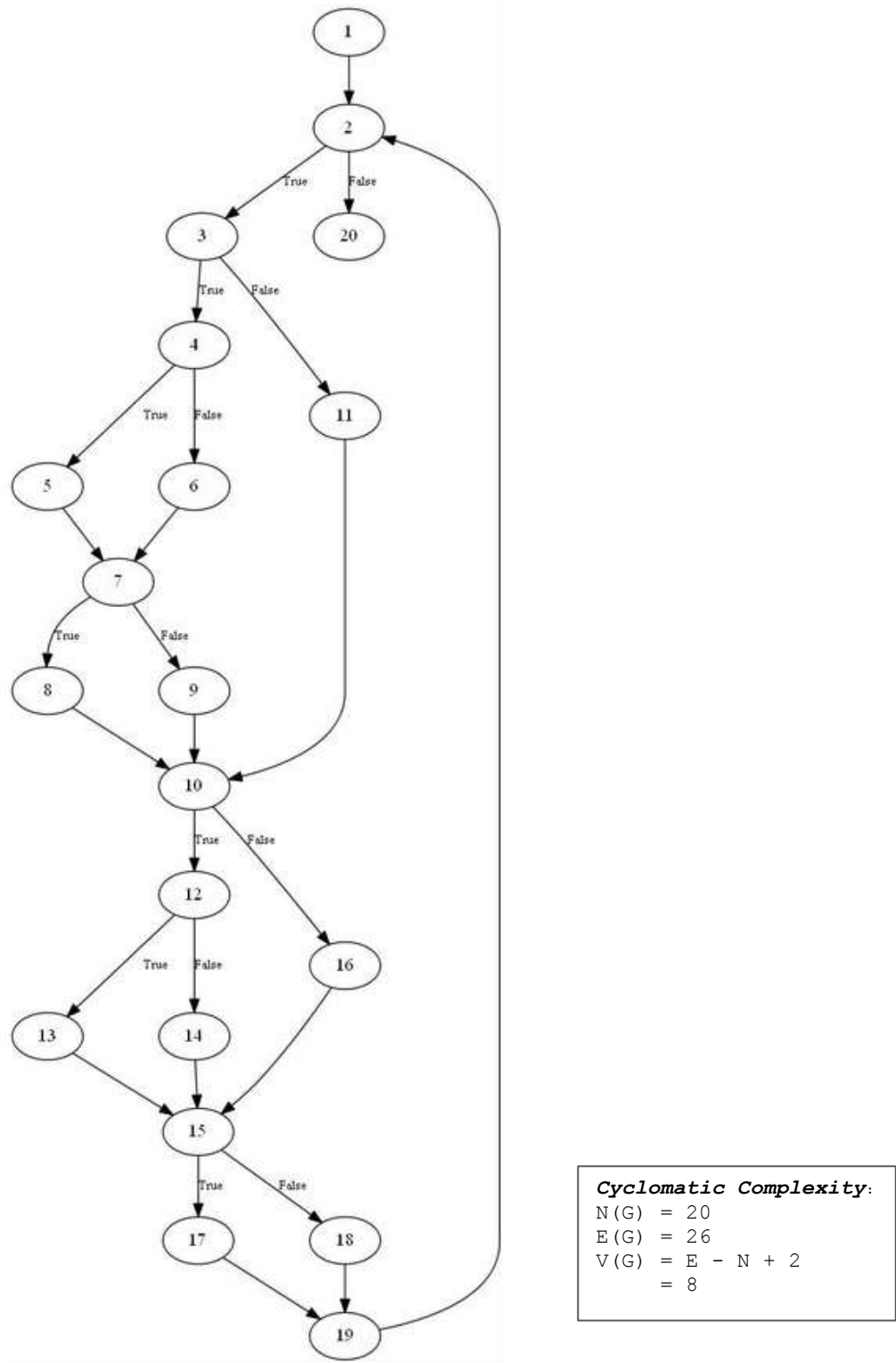
Lampiran 30 Hasil instrumentasi fG2011

1	Node 1	function [traversedPath,pop] = flexGong2011(depop)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		[px,py]=size(depop);
5		% instrument Branch # 1
6		traversedPath = [traversedPath '2 '];
7	Node 2	for i=1:px % Branch # 1
8		q=1;
9		p=depop(i,:);
10		lex_compat=p(1);
11		C_plus_plus=p(2);
12		fulltbl=p(3);
13		csize =p(4);
14		unspecified=p(5);
15		fullspd=p(6);
16		C_plus=p(7);
17		d(1,1)=2;
18		d(1,1)=1-1.001^(-d(1,1));
19		d(1,2)=lex_compat;
20		d(1,2)=1-1.001^(-d(1,2));
21		d(1,3)=0;
22		d(2,1)=2;
23		d(2,1)=1-1.001^(-d(2,1));
24		d(2,2)=C_plus_plus;
25		d(2,2)=1-1.001^(-d(2,2));
26		d(2,3)=0;
27		d(3,1)=2;
28		d(3,1)=1-1.001^(-d(3,1));
29		d(3,2)=fulltbl;
30		d(3,2)=1-1.001^(-d(3,2));
31		d(3,3)=0;
32		d(4,1)=abs(csize-unspecified)+2;
33		d(4,1)=1-1.001^(-d(4,1));
34		d(4,2)=2;
35		d(4,2)=1-1.001^(-d(4,2));
36		d(4,3)=0;
37		d(5,1)=2;
38		d(5,1)=1-1.001^(-d(5,1));
39		d(5,2)=fullspd;
40		d(5,2)=1-1.001^(-d(5,2));
41		d(5,3)=0;
42		d(6,1)=2;
43		d(6,1)=1-1.001^(-d(5,1));
44		d(6,2)=C_plus;
45		d(6,2)=1-1.001^(-d(5,2));
46		d(5,3)=0;
47		u=3*ones(1,6);
48		traversedPath = [traversedPath '(T) '];
49		% instrument Branch # 2
50		traversedPath = [traversedPath '3 '];
51	Node 3	if (lex_compat ~= 0) % Branch # 2

52		d(1,1)=0;
53		u(1)=1;
54		traversedPath = [traversedPath '(T) '];
55		% instrument Branch # 3
56		traversedPath = [traversedPath '4 '];
57	Node 4	if (C_plus_plus ~= 0) % Branch # 3
58		traversedPath = [traversedPath '(T) '];
59		traversedPath = [traversedPath '5 '];
60	Node 5	flexerror = 'Can not use -+ with -l option';
61		d(2,1)=0;
62		else
63		traversedPath = [traversedPath '(F) '];
64		traversedPath = [traversedPath '6 '];
65	Node 6	d(2,2)=0;
66		end
67		% instrument Branch # 4
68		traversedPath = [traversedPath '7 '];
69	Node 7	if (fulltbl ~= 0) % Branch # 4
70		traversedPath = [traversedPath '(T) '];
71		traversedPath = [traversedPath '8 '];
72	Node 8	flexerror='Can not use -f or -F with -l option';
73		d(3,1)=0;
74		else
75		traversedPath = [traversedPath '(F) '];
76		traversedPath = [traversedPath '9 '];
77	Node 9	d(3,2)=0;
78		end
79		else
80		traversedPath = [traversedPath '(F) '];
81		traversedPath = [traversedPath '11 '];
82	Node 11	d(1,2)=0;
83		end
84		% instrument Branch # 5
85		traversedPath = [traversedPath '10 '];
86	Node 10	if (csize == unspecified) % Branch # 5
87		d(4,1)=0;
88		traversedPath = [traversedPath '(T) '];
89		% instrument Branch # 6
90		traversedPath = [traversedPath '12 '];
91	Node 12	if (fullspd ~= 0) % Branch # 6
92		traversedPath = [traversedPath '(T) '];
93		traversedPath = [traversedPath '13 '];
94	Node 13	csize = 'DEFAULT_CSIZE';
95		d(5,1)=0;
96		else
97		traversedPath = [traversedPath '(F) '];
98		traversedPath = [traversedPath '14 '];
99	Node 14	d(5,2)=0;
100		csize = csize;
101		end
102		else
103		traversedPath = [traversedPath '(F) '];
104		traversedPath = [traversedPath '16 '];
105	Node 16	d(4,2)=0;
106		end
107		% instrument Branch # 7
108		traversedPath = [traversedPath '15 '];
109	Node 15	if (C_plus ~= 0) % Branch # 7
110		traversedPath = [traversedPath '(T) '];
111		traversedPath = [traversedPath '17 '];
112	Node 17	suffix='cc';
113		d(6,1)=0;
114		else
115		traversedPath = [traversedPath '(F) '];
116		traversedPath = [traversedPath '18 '];
117	Node 18	d(6,2)=0;
118		suffix='c';
119		outfilename = 'outfile_path';
120	Node 19	end
121		traversedPath = [traversedPath '19 '];
122		pop(i,:) = p;

123		traversedPath = [traversedPath '2 '];
124		end
125		traversedPath = [traversedPath '(F) '];
126		traversedPath = [traversedPath '20 '];
127	Node 20	end

Lampiran 31 CFG dan *Cyclomatic Complexity* fg2011



Lampiran 32 Semua kemungkinan jalur fG2011

- 1 2 (T) 3 (T) 4 (T) 5 7 (T) 8 10 (T) 12 (T) 13 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (T) 8 10 (T) 12 (T) 13 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (T) 8 10 (T) 12 (F) 14 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (T) 8 10 (T) 12 (F) 14 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (T) 8 10 (F) 16 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (T) 8 10 (F) 16 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (F) 9 10 (T) 12 (T) 13 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (F) 9 10 (T) 12 (T) 13 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (F) 9 10 (T) 12 (F) 14 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (F) 9 10 (T) 12 (F) 14 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (F) 9 10 (F) 16 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (T) 5 7 (F) 9 10 (F) 16 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (T) 8 10 (T) 12 (T) 13 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (T) 8 10 (T) 12 (T) 13 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (T) 8 10 (T) 12 (F) 14 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (T) 8 10 (T) 12 (F) 14 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (T) 8 10 (F) 16 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (T) 8 10 (F) 16 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (F) 9 10 (T) 12 (T) 13 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (F) 9 10 (T) 12 (T) 13 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (F) 9 10 (T) 12 (F) 14 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (F) 9 10 (T) 12 (F) 14 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (F) 9 10 (F) 16 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (T) 4 (F) 6 7 (F) 9 10 (F) 16 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (F) 11 10 (T) 12 (T) 13 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (F) 11 10 (T) 12 (T) 13 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (F) 11 10 (T) 12 (F) 14 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (F) 11 10 (T) 12 (F) 14 15 (F) 18 19 2 (F) 20
- 1 2 (T) 3 (F) 11 10 (F) 16 15 (T) 17 19 2 (F) 20
- 1 2 (T) 3 (F) 11 10 (F) 16 15 (F) 18 19 2 (F) 20
- 1 2 (F) 20

Lampiran 33 Kode program fmH2015

1	Node 1	function branchVal = fitnessMiniMaxi(branchNo, predicate)
2		k = 1; % the smallest step for integer
3	Node 2	switch (branchNo)
4	Node 3	case 1,
5		% branch #1: (idx <= numLength)
6		branchVal = predicate(1) - predicate(2);
7	Node 4	case 2,
8		% branch #2: (maxi < num(idx))
9		branchVal = predicate(1) - predicate(2);
10	Node 5	case 3,
11		% branch #3: (mini > num(idx))
12		branchVal = predicate(2) - predicate(1);
13		end
14	Node 6	if ((branchNo == 2) (branchNo == 3)),
15	Node 7	if (branchVal < 0)
16	Node 8	branchVal = branchVal - k;
17		else
18	Node 9	branchVal = branchVal + k;

19		end
20		end
21	Node 10	end

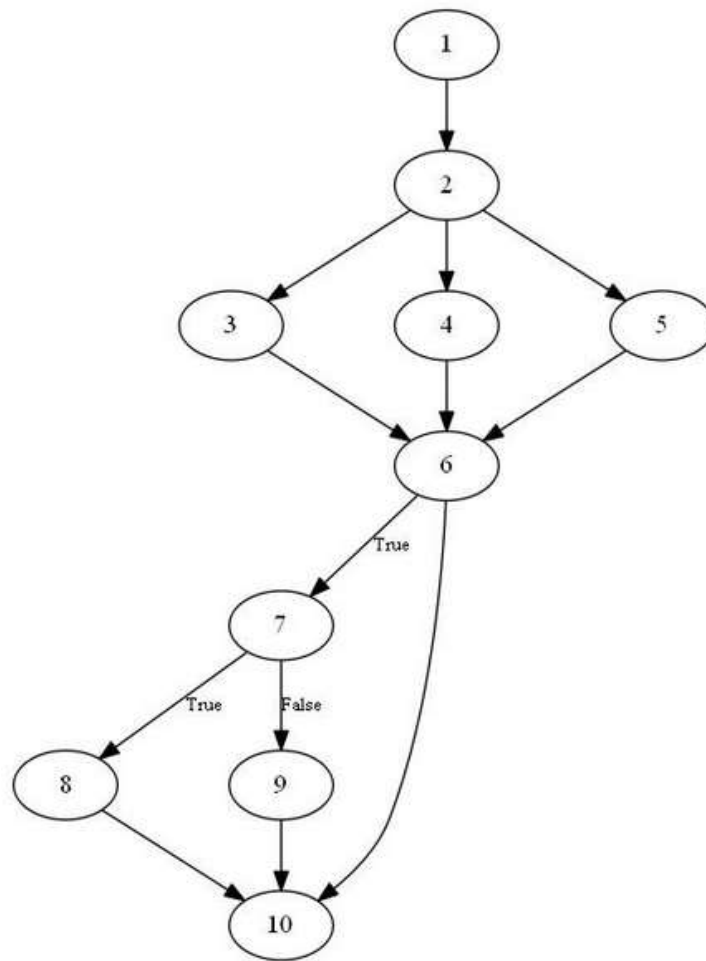
Lampiran 34 Hasil instrumentasi fmH2015

1	Node 1	function [traversedPath,branchVal] = fitnessMiniMaxi(branchNo, predicate)
2		traversedPath = [];
3		traversedPath = [traversedPath '1 '];
4		k = 1; % the smallest step for integer
5		traversedPath = [traversedPath '2 '];
6	Node 2	switch (branchNo)
7		case 1,
8	Node 3	traversedPath = [traversedPath '3 '];
9		% branch #1: (idx <= numLength)
10		branchVal = predicate(1) - predicate(2);
11		case 2,
12	Node 4	traversedPath = [traversedPath '4 '];
13		% branch #2: (maxi < num(idx))
14		branchVal = predicate(1) - predicate(2);
15		case 3,
16	Node 5	traversedPath = [traversedPath '5 '];
17		% branch #3: (mini > num(idx))
18		branchVal = predicate(2) - predicate(1);
19		end
20		% instrument Branch # 1
21		traversedPath = [traversedPath '6 '];
22	Node 6	if ((branchNo == 2) (branchNo == 3)),
23		traversedPath = [traversedPath '(T) '];
24		% instrument Branch # 2
25		traversedPath = [traversedPath '7 '];
26	Node 7	if (branchVal < 0)
27		traversedPath = [traversedPath '(T) '];
28		traversedPath = [traversedPath '8 '];
29	Node 8	branchVal = branchVal - k;
30		else
31		traversedPath = [traversedPath '(F) '];
32		traversedPath = [traversedPath '9 '];
33	Node 9	branchVal = branchVal + k;
34		end
35		end
36		end
37		traversedPath = [traversedPath '10 '];
38	Node 10	end

Lampiran 35 CFG dan *Cyclomatic Complexity* fmH2015

Cyclomatic Complexity:

$$\begin{aligned}
 N(G) &= 10 \\
 E(G) &= 13 \\
 V(G) &= E - N + 2 \\
 &= 5
 \end{aligned}$$



Lampiran 36 Semua kemungkinan jalur fmH2015

- 1 2 3 6 (T) 7 (T) 8 10
- 1 2 3 6 (T) 7 (F) 9 10
- 1 2 3 6 10
- 1 2 4 6 (T) 7 (T) 8 10
- 1 2 4 6 (T) 7 (F) 9 10
- 1 2 4 6 10
- 1 2 5 6 (T) 7 (T) 8 10
- 1 2 5 6 (T) 7 (F) 9 10
- 1 2 5 6 10

RIWAYAT HIDUP

Raden Asri Ramadhina Fitriani dilahirkan di Bogor Jawa Barat pada tanggal 9 Maret 1993. Penulis merupakan anak ketiga dari pasangan Raden Achmad Mulyadi dan Tini Hertini. Penulis mempunyai dua orang saudara, yaitu Raden Indra Yuga Pratama dan Raden Agung Yuga Dwitama

Penulis menyelesaikan pendidikan Sekolah Menengah Atas pada tahun 2011 di SMA Negeri 10 Bogor. Penulis menyelesaikan pendidikan perguruan tinggi di Program Diploma Institut Pertanian Bogor melalui jalur Undangan Seleksi Masuk IPB (USMI) di program keahlian Manajemen Informatika pada tahun 2011 hingga tahun 2014 dengan mendapatkan penghargaan sebagai lulusan terbaik.

Penulis melanjutkan pendidikan tingkat Sarjana di Alih Jenis Departemen Ilmu Komputer Institut Pertanian Bogor. Selama mengikuti perkuliahan, penulis bekerja di Direktorat Integrasi Data dan Sistem Informasi Institut Pertanian Bogor semenjak lulus pendidikan tinggi di Program Diploma. Selama melakukan penelitian, penulis merangkap menjadi asisten praktikum Sistem Cerdas di Alih Jenis Departemen Ilmu Komputer Institut Pertanian Bogor pada tahun ajaran 2017/2018 Genap.