

**ILMU KOMPUTER**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**INSTITUT PERTANIAN BOGOR**

**BOGOR**

**2018**

**RADEN ASRI RAMADHINA FITRIANI**

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

**PERNYATAAN MENGENAI SKRIPSI DAN  
SUMBER INFORMASI SERTA PELIMPAHAN HAK CIPTA**

Dengan ini saya menyatakan bahwa skripsi berjudul Instrumentasi Kode Program Secara Otomatis Untuk *Basis 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 disertasi ini.

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

Bogor, Maret 2018

*Raden Asri Ramadhina Fitriani*

NIM G64154007

**ABSTRAK**

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

Pengujian adalah serangkaian proses yang dirancang untuk memastikan sebuah perangkat lunak melakukan apa yang seharusnya dilakukan. *Basis 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. Kumar dan Mishra (2016) mengatakan bahwa pengujian perangkat lunak menggunakan hampir 60% dari total biaya pengembangan perangkat lunak. Sehingga 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#.

**ABSTRACT**

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

Testing is series of processes designed to ensure a software does whar it’s supposed to do. Basis 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 cmplex softwareas a whole will take a long time and require a lot of human resources. Kumar and Mishra (2016) say that software testing uses nearly 60% of total software development costs. Auromating 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 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:

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

**RADEN ASRI RAMADHINA FITRIANI**

**DEPARTEMEN ILMU KOMPUTER**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**INSTITUT PERTANIAN BOGOR**

**BOGOR**

**2018**

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

Judul Skripsi : Instrumentasi Kode Program Secara Otomatis untuk *Basis 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 *Basis 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, Maret 2018

*Raden Asri Ramadhina Fitriani*

**DAFTAR ISI**

DAFTAR TABEL vi

DAFTAR GAMBAR vi

DAFTAR LAMPIRAN vi

[PENDAHULUAN 9](#_Toc509315380)

[Latar Belakang 9](#_Toc509315381)

[Perumusan Masalah 10](#_Toc509315382)

[Tujuan Penelitian 10](#_Toc509315383)

[Ruang Lingkup Penelitian 10](#_Toc509315384)

[Manfaat Penelitian 10](#_Toc509315385)

[METODE 11](#_Toc509315386)

[Analisis 11](#_Toc509315387)

[Perancangan 11](#_Toc509315388)

[Implementasi 15](#_Toc509315389)

[Testing 15](#_Toc509315390)

[HASIL DAN PEMBAHASAN 15](#_Toc509315391)

[Analisis 15](#_Toc509315392)

[Perancangan 16](#_Toc509315393)

[Implementasi 18](#_Toc509315394)

[Testing 27](#_Toc509315395)

[SIMPULAN DAN SARAN 31](#_Toc509315396)

[Simpulan 31](#_Toc509315397)

[Saran 31](#_Toc509315398)

[DAFTAR PUSTAKA 31](#_Toc509315399)

[LAMPIRAN 33](#_Toc509315400)

[RIWAYAT HIDUP 58](#_Toc509315401)

**DAFTAR TABEL**

[Contoh program uji 16](#_Toc509315326)

[Perbandingan *adjacency list* manual dan menggunakan aplikasi 27](#_Toc509315327)

[Perbandingan waktu eksekusi secara menual dan menggunakan aplikasi 30](#_Toc509315328)

**DAFTAR GAMBAR**

[Tahapan penelitian 11](#_Toc509315329)

[Arsitektur sistem 12](#_Toc509315330)

[Notasi Control Flow Graph (CFG) 14](#_Toc509315331)

[Perancangan *Class Diagram* 17](#_Toc509315332)

[Perancangan antarmuka *form awal* 17](#_Toc509315333)

[Perancangan antarmuka hasil dari proses yang telah dilakukan 18](#_Toc509315334)

[Potongan hasil penguraian kode program tA2008 ke dalam format XML 20](#_Toc509315335)

[Hasil *nodes* yang terbentuk dari tA2008 21](#_Toc509315336)

[Representasi *Object* *Graph* tA2008 dalam bentuk 21](#_Toc509315337)

[Representasi tA2008 dalam bahasa dot 22](#_Toc509315338)

[CFG tA2008 23](#_Toc509315339)

[Hasil instrumentasi tA2008 24](#_Toc509315340)

[Hasil ekseskusi kode program tA2008 yang sudah diinstrumentasi 25](#_Toc509315341)

[Tampilan halaman awal apikasi 25](#_Toc509315342)

[Tampilan hasil pembangkitan 26](#_Toc509315343)

**DAFTAR LAMPIRAN**

[Kode program mmA2008 33](#_Toc509315344)

[Hasil instrumentasi mmA2008 33](#_Toc509315345)

[CFG dan *Cyclomatic Complexity* mmA2008 34](#_Toc509315346)

[Semua kemungkinan jalur mmA2008 34](#_Toc509315347)

[Kode program iA2008 34](#_Toc509315348)

[Hasil instrumentasi iA2008 35](#_Toc509315349)

[CFG dan *Cyclomatic Complexity* iA2008 35](#_Toc509315350)

[Semua kemungkinan jalur iA2008 36](#_Toc509315351)

[Kode program binA2008 36](#_Toc509315352)

[Hasil instrumentasi binA2008 36](#_Toc509315353)

[CFG dan *Cyclomatic Complexity* binA2008 37](#_Toc509315354)

[Semua kemungkinan jalur binA2008 38](#_Toc509315355)

[Kode program bubA2008 39](#_Toc509315356)

[Hasil instrumentasi bubA2008 39](#_Toc509315357)

[CFG dan *Cyclomatic Complexity* bubA2008 40](#_Toc509315358)

[Semua kemungkinan jalur bubA2008 40](#_Toc509315359)

[Kode program gA2008 40](#_Toc509315360)

[Hasil instrumentasi gA2008 41](#_Toc509315361)

[CFG dan *Cyclomatic Complexity* gA2008 41](#_Toc509315362)

[Semua kemungkinan jalur gA2008 42](#_Toc509315363)

[Kode program eB2002 42](#_Toc509315364)

[Hasil instrumentasi eB2002 43](#_Toc509315365)

[CFG dan *Cyclomatic Complexity* eB2002 45](#_Toc509315366)

[Semua kemungkinan jalur eB2002 46](#_Toc509315367)

[Kode program qB2002 47](#_Toc509315368)

[Hasil instrumentasi qB2002 47](#_Toc509315369)

[CFG dan *Cyclomatic Complexity* qB2002 48](#_Toc509315370)

[Semua kemungkinan jalur qB2002 48](#_Toc509315371)

[Kode program fG2011 49](#_Toc509315372)

[Hasil instrumentasi fG2011 51](#_Toc509315373)

[CFG dan *Cyclomatic Complexity* fG2011 53](#_Toc509315374)

[Semua kemungkinan jalur fG2011 55](#_Toc509315375)

[Kode program fmH2014 55](#_Toc509315376)

[Hasil instrumentasi fmH2014 56](#_Toc509315377)

[CFG dan *Cyclomatic Complexity* fmH2014 56](#_Toc509315378)

[Semua kemungkinan jalur fmH2014 57](#_Toc509315379)

# 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. Basis 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. Sehingga 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 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.

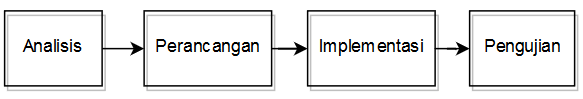
## Manfaat Penelitian

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 menunjukan tahapan proses tersebut.



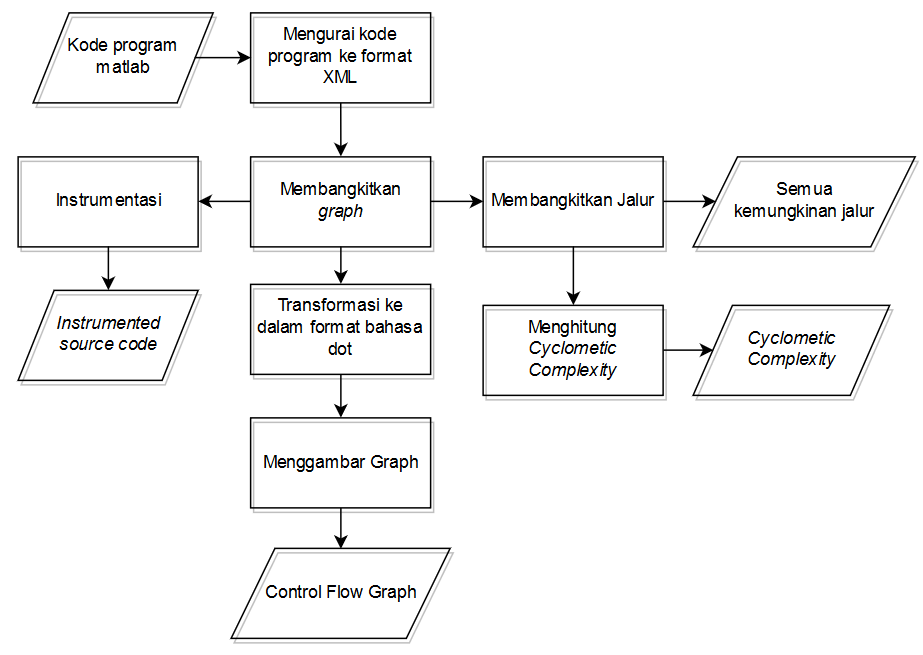
Gambar 1 Tahapan penelitian

## **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 berapa contoh program yang akan digunakan dalam penelitian. Contoh program yang akan digunakan pada penelitian ini memiliki perintah struktur kontrol.

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

**Mengurai Kode Progam ke Format XML**

Penguraian kode program matlab dilakukan dengan menggunakan *library*  MATLAB-PARSER. Ketika terdapat kesalahan pada kode program, *library*  ini akan mengembalikan pesan *error.* 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 sebegai informasi tambahan dari setiap elemen (Hartwell 2017).

**Membangkitkan *Graph***

Setiap elemen dalam file XML tersebut akan ditelusuri satu persatu yang termasuk struktur kontrol di dalam bahasa matlab. Sehingga terbentuklah sebuah objek *graph* yang terdiri dari sekumpulan *node* dan *edge*.

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 struktur kontrol pada program akan menjadi *node* baru. Setiap *node* berisi informasi nomor baris dan nomor kolom yang akan digunakan untuk melakukan instrumentasi.

**Membangkitkan Jalur**

*Basis 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).

Metode ini terbagi menjadi 4 tahapan, yaitu:

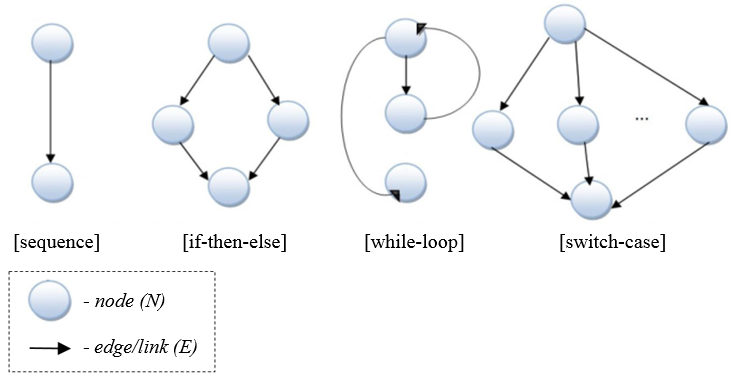
1. Menggambarkan jalur dalam bentuk *Control Flow Graph* (CFG)
2. Menghitung *cyclomatic complexity*
3. Memilih satu set jalur dasar
4. Membangkitkan data uji untuk setiap jalur dasar

**Transformasi ke Dalam Format Bahasa Dot**

*Graph* yang sudah terbentuk akan ditransformasikan ke dalam bentuk format bahasa permrograman dot. Bahasa dot adalah bahasa yang digunakan untuk mengambar *graph* berarah. Bahasa ini dapat mendeskripsikan 3 macam objek, yaitu *graph, nodes,* dan *edges* (Ganser *et all* 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 all* 2003).

**Menghitung *Cyclometic Complexity***

*Cyclomatic complexity* merupakan suatu sistem pengukuran yang ditemukan oleh Watson dan McCabe untuk menentukan banyaknya *independent path* dan menunjukan 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 seperti awal setiap perintah, sebelum atau sesudah kondisi terpenuhi atau tidak. Dalam pengujian 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 in 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. Graphvizmerupakan *library* yang dapat digunakan untuk menvisualisasi 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. 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 dilaukan 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.

# 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.

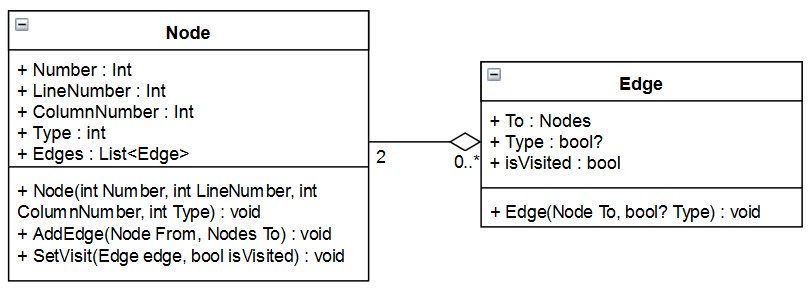
Tabel 1 Contoh program uji

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Program Uji** | **Nama** | **Deskripsi** |
| 1 | Triangle Ahmed | tA2008 | Menentukan tipe dari segitiga apakah termasuk *equilateral, isosceles, scalene,* atau *not triangle* |
| 2 | Minimaxi Ahmed | mmA2008 | Menentukan nilai minimal dan maksimal dari inputan berupa bilangan dalam *array* |
| 3 | Binary Ahmed | binA2008 | mencari indeks sebuah bilangan dalam array dengan mengembalikan indeks jika ditemukan dan tidak jika tidak ditemukan. |
| 4 | Bubble Ahmed | bubA2008 | Mengurutkan bilangan dalam *array* menggunakan metode *bubble sort* |
| 5 | Quotient Bueno | qB2002 | Menghitung hasil bagi dan sisa hasil bagi dari dua buah bilangan bulat positif |
| 6 | Fitness Minimaxi Hermadi | fmH2014 | Menghitung fungsi *fitness* dari fungsi minimaxiAhmed2008 |
| 7 | Insertion Ahmed | iA2008 | Mengurutkan bilangan dalam *array* menggunakan metode *insertion sort* |
| 8 | Gcd Ahmed | gA2008 | Menghitung *GCD* atau pembagi dua bilangan terbesar |
| 9 | Expint Bueno | eB2002 | Fungsi exponensial yang dapat memproses bilangan *integer* dan *float* |
| 10 | Flex Gong | fG2011 | Sebuah utilitas unix yang diambil dari situs GNU |

## **Perancangan**

**Perancangan *Class Diagram***

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

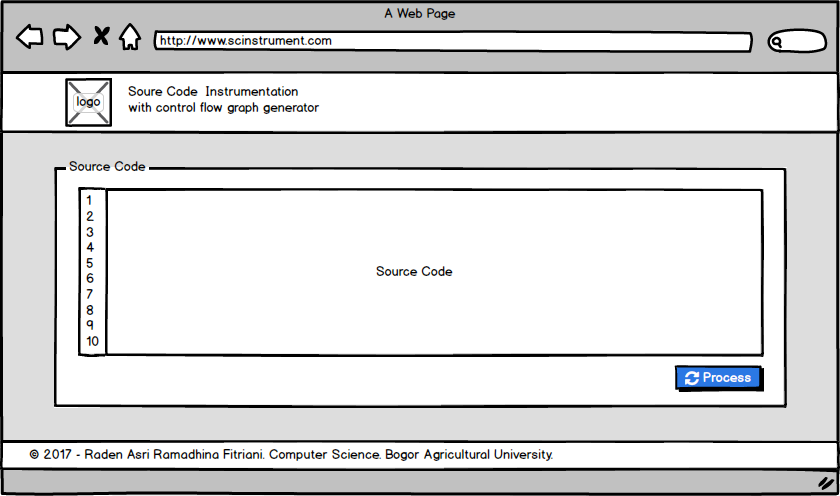


Gambar 4 Perancangan *Class Diagram*

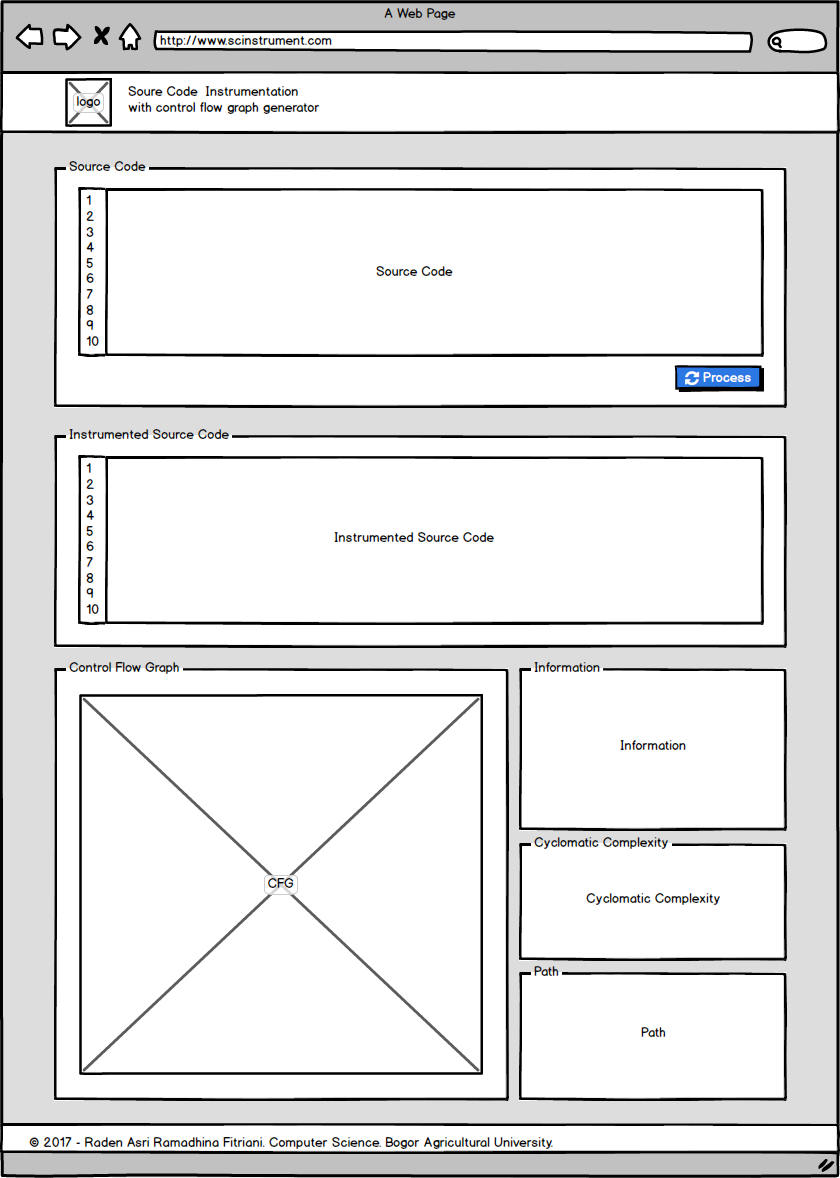
Dalam sebuah *class node* terdapat informasi nomor *node*, nomor baris dan nomor kolom dari kode program, dan tipe dari perintah tersebut apakah termasuk percabangan, pengulangan, perintah biasa, atau akhir dari sebuah perintah. Selain itu, terdapat list *edge* yang berisi *node* tujuan dan tipe dari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

Berikut merupakan kode program tA2008, yaitu untuk mencari jenis dari segitiga jika diketahui panjang dari setiap sisinya.

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

**Mengurai Kode Progam 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 7 menunjukkan potongan hasil penguraian kode program tA2008 ke dalam format XML. Potongan kode XML yang terlihat pada Gambar 7 menunjukkan hasil penguraian dari kode program pada baris ke 6 sampai baris ke 7.



Gambar 7 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***

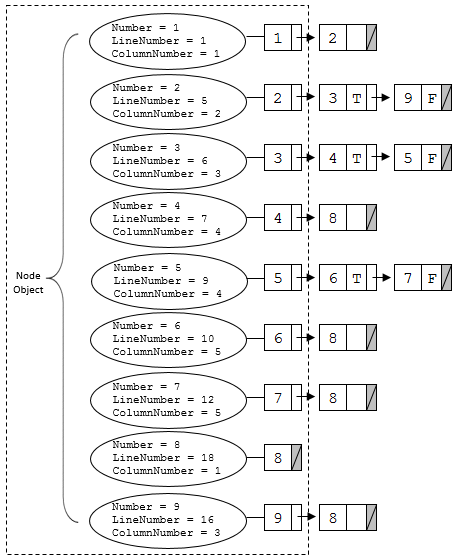
Salah satu cara untuk membaca dan menulis dokumen XML pada *framework* .NET dan C# yaitu dengan menggunakan kelas *XMLDocument* yang terdapat dalam *namespace System.XML*. Setiap elemen XML yang merupakan struktur kontrol pada program akan menjadi *node* baru. Setiap *node* berisi informasi nomor baris 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 8. *Node* 1 dibentuk pada awal kode program sebagai inisialisasi. *Node* ditambahkan ketika bertemu dengan perintah yang termasuk ke dalam struktur kontrol 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  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 9**  **Node 8** | function type = triangle(sideLengths)  A = sideLengths(1); % First side  B = sideLengths(2); % Second side  C = sideLengths(3); % Third side  if ((A+B > C) && (B+C > A) && (C+A > B))  if ((A ~= B) && (B ~= C) && (C ~= A))  type = 'Scalene';  else  if (((A == B) && (B ~= C)) || ((B == C) && (C ~= A)) || ((C == A) && (A ~= B)))  type = 'Isosceles';  else  type = 'Equilateral';  end  end  else  type = 'Not a triangle';  end  end |

Gambar 8 Hasil *nodes* yang terbentuk dari tA2008

Representasi objek dari kelas *graph* yang terbentuk dari kode program tA2008 dapat dilihat pada Gambar 9. *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 9 Representasi *Object* *Graph* tA2008 dalam bentuk

**Membangkitkan Jalur**

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. Berikut merupakan semua kemungkinan jalur yang akan dilalui ketika diberikan suatu inputan 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**

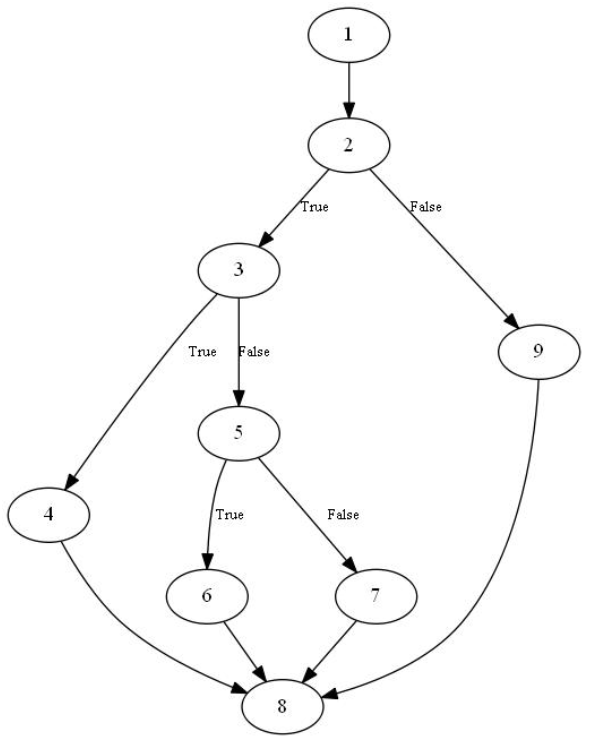
Transformasi ke dalam format bahasa dot dilakukan dengan cara menelusuri objek *graph* yang sudah dibangun sebelumnya. Yang didefinisikan dalam bahasa dot adalah *edge* yang terdapat pada *graph* yang dibangun. Seperti yang dapat dilihat pada Gambar 10, jumlah baris sebanyak jumlah *edge* pada objek *graph* yang telah didefinisikan sebelumnya.

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

Gambar 10 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 11.



Gambar 11 CFG tA2008

**Menghitung *Cyclometic 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 9 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.

Nodes(N) = 9

Edges(E) = 11

V(G) = E - N + 2

         = 4

**Instrumentasi**

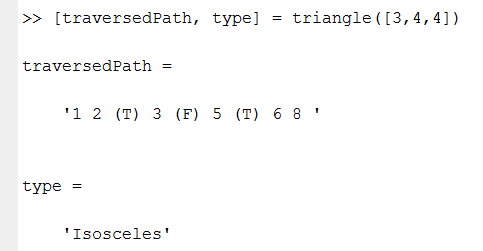
Instrumentasi dilakukan dengan cara menambahkan dulu variabelkeluaran bernama *traversedPath*. Variabelin digunakan untuk menyimpan informasi *node* mana saja yang dilalui ketika diberikan inputan dengan nilai tertentu. Dan menyimpan informasi pilihan yang dilalui ketika ditemukan cabang yang terdapat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

Hasil kode program yang telah diinstrumentasi dapat dilihat pada Gambar 12. 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 13.

Sehingga ketika kode program hasil instrumentasi dijalankan dengan inputan tertentu akan menghasilkan keluaran variabel *traversedPath* dan *type* seperti yang dapat dilihat pada Gambar 13. Misalkan inputan adalah 3, 4, dan 4 akan menghasilkan *isosceles* dan dapat diketahui bagaimana cara menghasilkan keluaran tersebut dari *traversedPath.*

|  |  |
| --- | --- |
| 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  27  28  29  30  31  32  33  34  35  36  37 | function [traversedPath,type] = triangle(sideLengths)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  A = sideLengths(1); % First side  B = sideLengths(2); % Second side  C = sideLengths(3); % Third side  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  if ((A+B > C) && (B+C > A) && (C+A > B))  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '3 ' ];  if ((A ~= B) && (B ~= C) && (C ~= A))  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '4 ' ];  type = 'Scalene';  else  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 3  traversedPath = [traversedPath '5 ' ];  if (((A == B) && (B ~= C)) || ((B == C) && (C ~= A)) || ((C == A) && (A ~= B)))  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '6 ' ];  type = 'Isosceles';  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '7 ' ];  type = 'Equilateral';  end  end  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '9 ' ];  type = 'Not a triangle';  end  traversedPath = [traversedPath '8 ' ];  end |

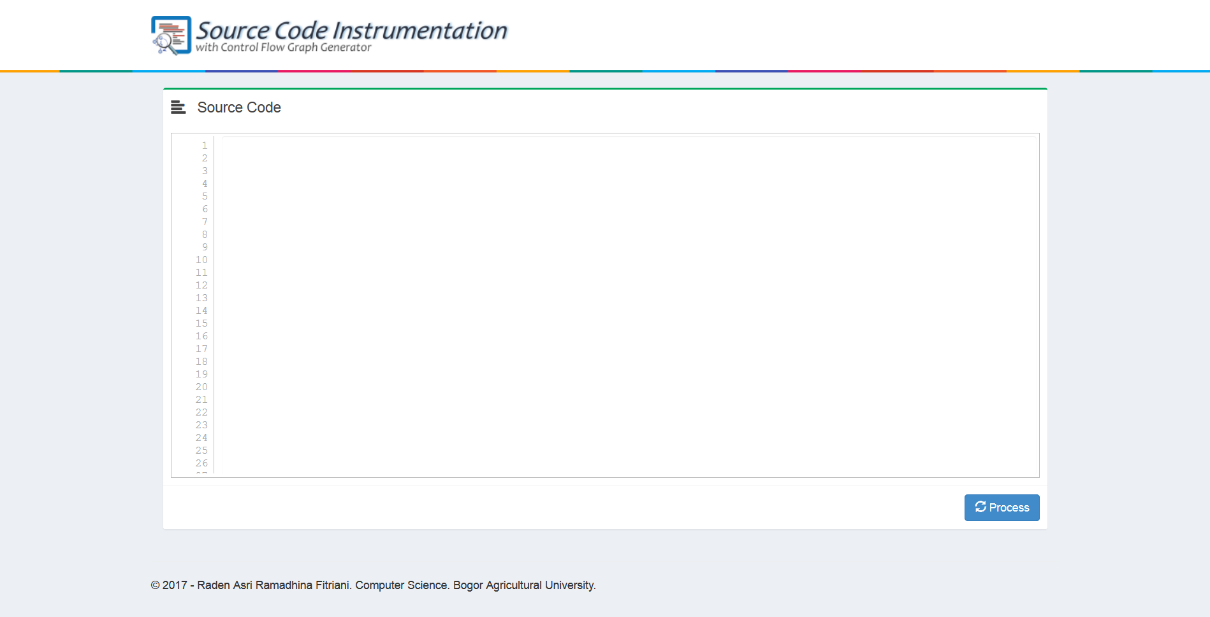
Gambar 12 Hasil instrumentasi tA2008



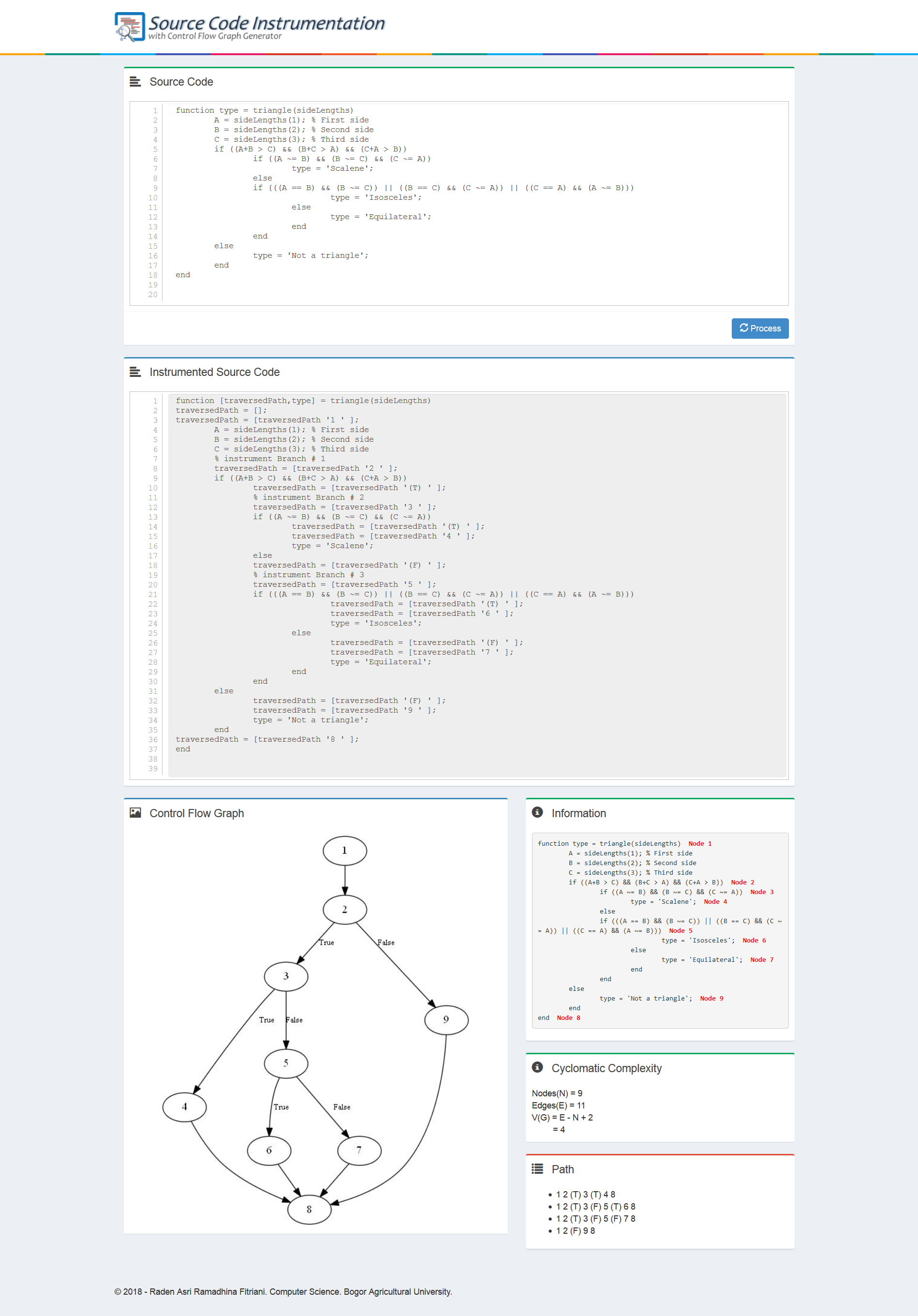
Gambar 13 Hasil ekseskusi kode program tA2008 yang sudah diinstrumentasi

**Implementasi Antarmuka**

Halaman awal dari aplikasi dapat dilihat pada Gambar 14 dimana hanya terdapat satu text area untuk memasukkan kode program yang akan di proses. Tampilan hasil pembangkitan dapat dilihat pada Gambar 15.



Gambar 14 Tampilan halaman awal apikasi



Gambar 15 Tampilan hasil pembangkitan

## 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* 4,5,6,7,8,9 menjadi *node* 9,4,5,6,7,8 di *graph* yang dibangun menggunakan aplikasi.

Tabel 2 Perbandingan *adjacency list* manual dan menggunakan aplikasi

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Program** | **Adjacency List** | **Adjacency List** |
| 1 | tA2008 | * 1 -> 2 * 2 -> 3 -> 4 * 3 -> 5 -> 6 * 4 -> 9 * 5 -> 9 * 6 -> 7 -> 8 * 7 -> 9 * 9 * 8 -> 9 | * 1 -> 2 * 2 -> 3 -> 9 * 3 -> 4 -> 5 * 4 -> 8 * 5 -> 6 -> 7 * 6 -> 8 * 7 -> 8 * 8 * 9 -> 8 |
| 2 | mmA2008 | * 1 -> 2 * 2 -> 3 -> 8 * 3 -> 4 -> 5 * 4 -> 5 * 5 -> 6 -> 7 * 6 -> 7 * 7 -> 2 * 8 | * 1 -> 2 * 2 -> 3 -> 8 * 3 -> 4 -> 5 * 4 -> 5 * 5 -> 6 -> 7 * 6 -> 7 * 7 -> 2 * 8 |
| 3 | iA2008 | * 1 -> 2 * 2 -> 3 -> 6 * 3 -> 4 -> 5 * 4 -> 3 * 5 -> 2 * 6 | * 1 -> 2 * 2 -> 3 -> 6 * 3 -> 4 -> 5 * 4 -> 3 * 5 -> 2 * 6 |
| 4 | binA2008 | * 1 -> 2 * 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 | * 1 -> 2 * 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 | * 1 -> 2 * 2 -> 3 -> 8 * 3 -> 4 -> 7 * 4 -> 5 -> 6 * 5 -> 6 * 6 -> 3 * 7 -> 2 * 8 | * 1 -> 2 * 2 -> 3 -> 8 * 3 -> 4 -> 7 * 4 -> 5 -> 6 * 5 -> 6 * 6 -> 3 * 7 -> 2 * 8 |
| 6 | gA2008 | * 1 -> 2 * 2 -> 3 -> 4 * 3 -> 9 * 4 -> 5 -> 9 * 5 -> 6 -> 7 * 6 -> 8 * 7 -> 8 * 8 -> 4 * 9 | * 1 -> 2 * 2 -> 3 -> 4 * 3 -> 9 * 4 -> 5 -> 9 * 5 -> 6 -> 7 * 6 -> 8 * 7 -> 8 * 8 -> 4 * 9 |
| 7 | eB2002 | * 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 | * 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 |
| 8 | qB2002 | * 1 -> 2 * 2 -> 3 -> 10 * 3 -> 4 -> 10 * 4 -> 5 -> 6 * 5 -> 4 * 6 -> 7 -> 10 * 7 -> 8 -> 9 * 8 -> 9 * 9 -> 6 * 10 | * 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 | * 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 | * 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 | fmH2014 | * 1 -> 2 * 2 -> 3 -> 4 -> 5 * 3 -> 6 * 4 -> 6 * 5 -> 6 * 6 -> 7 -> 10 * 7 -> 8 -> 9 * 8 -> 10 * 9 -> 10 * 10 | * 1 -> 2 * 2 -> 3 -> 4 -> 5 * 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 dilaukan 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 1.64 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 9 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 1.47 detik.

Tabel 3 Perbandingan waktu eksekusi secara menual dan menggunakan aplikasi

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Nama Program** | **Waktu Eksekusi Aplikasi (detik)** | **Waktu Ekseskusi 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 | gA2008 | 0.21 | 492.93 | 293.01 | 392.97 |
| 7 | eB2002 | 0.22 | 686.59 | 632.27 | 659.43 |
| 8 | qB2002 | 0.16 | 222.25 | 280.86 | 251.56 |
| 9 | fmH2014 | 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.

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, Aldi. 2009. “Control Flow Graph Generator Documentation”. [Internet]. [Diunduh tanggal 19/7/2017 ]. Dapat diunduh dari: <http://eclipsefcg.sourceforge.net/Documentation.pdf.>

Arkeman, Y, Herdiyeni, Y, Hermadi, I, dan Laxmi, G F. 2014. *Algoritma Genetika Tujuan Jamak (MultiObjective Genetic Algorithm*. IPB Press.

Basu, A. 2015. *Software Quality Assurance, Testing and Metrics*. PHI Learning Privat Limited. [Internet]. [Diunduh tanggal 14/8/2017 ]. Dapat diunduh dari: <https://github.com/JamieDixon/GraphViz-C-Sharp-Wrapper>.

Dixon, J. 2013. “Graphviz.Net C# Wrapper”. [Internet]. [Diunduh tanggal 22/12/2017 ]. Dapat diunduh dari: <https://github.com/JamieDixon/GraphViz-C-Sharp-Wrapper>.

Ellson, J *et al.* 2003. “Graphviz and Dynagraph – Static and Dynamic Graph Drawing Tools”. [Internet]. [Diunduh tanggal 14/8/2017 ]. Dapat diunduh dari: <https://github.com/JamieDixon/GraphVizC-Sharp-Wrapper>.

Gansner, Emden R., Koutsofios, Eleftherios, dan North, Stephen. 2015. “Drawing graphs with dot”. [Internet]. [Diunduh tanggal 25/12/2017 ]. Dapat diunduh dari: <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.

Houcque, David. 2015. “Intoruction To MATLAB For Engineering Students”. [Internet]. [Diunduh tanggal 30/12/2017 ]. Dapat diunduh dari: <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 tanggal 20/8/2017]. Dapat diunduh dari: [http://www.sciencedirect.com/science/article/pii/%5C%5CS1877050916001277.](http://www.sciencedirect.com/)

Myers, G J, Sandler, C, dan Badgett, T. 2012. *The Art of Software Testing*. John Willey dan Sons, Inc, Hoboken, New York. [Internet]. [Diunduh tanggal 14/8/2017 ]. Dapat diunduh dari: <https://books.google.co.id/books>.

Suffos, S. 2015. “Matlab Parser”. [Internet]. [Diunduh tanggal 22/12/2017 ]. Dapat diunduh dari: [https//github.com/samuel-suffos/matlabparser.](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 tanggal 21/8/2017 ]. Dapat diunduh dari: <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 tanggal 14/8/2017 ]. Dapat diunduh dari: <http://www.mccabe.com/pdf/mccabe-nist235r.pdf.>

# LAMPIRAN

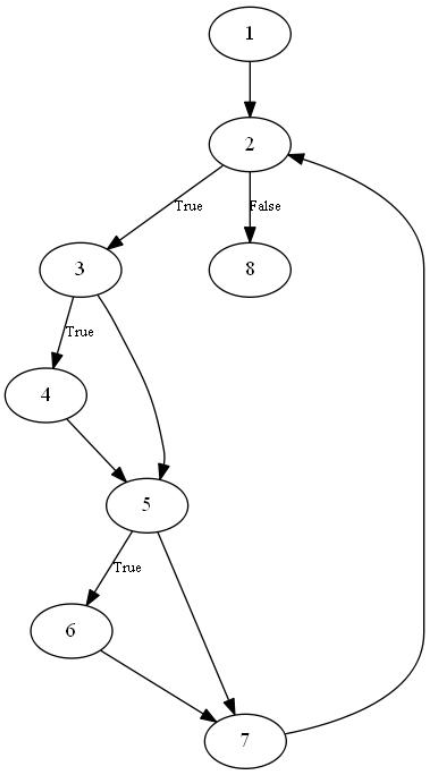
Lampiran 1 Kode program mmA2008

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

Lampiran 2 Hasil instrumentasi mmA2008

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8** | function [traversedPath,miniMaxi] = minimaxi(num)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  numLength = length(num);  mini = num(1);  maxi = num(1);  idx = 2;  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  while (idx <= numLength) % Branching #1  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '3 ' ];  if maxi < num(idx) % Branching #2  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '4 ' ];  maxi = num(idx);  end  % instrument Branch # 3  traversedPath = [traversedPath '5 ' ];  if mini > num(idx) % Branching #3  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '6 ' ];  mini = num(idx);  end  traversedPath = [traversedPath '7 ' ];  idx = idx+1;  traversedPath = [traversedPath '2 ' ];  end  miniMaxi = [mini maxi];  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '8 ' ];  end |

Lampiran 3 CFG dan *Cyclomatic Complexity* mmA2008



***Cyclomatic Complexity*:**

N(G) = 8

E(G) = 10

V(G) = E – N + 2

= 4

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

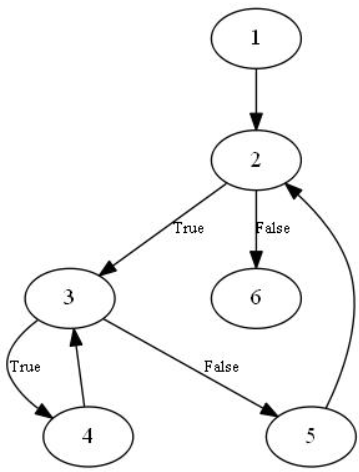
Lampiran 5 Kode program iA2008

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

Lampiran 6 Hasil instrumentasi iA2008

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6** | function [traversedPath,sortedArray] = insertion(anyArray)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  k = 1; % The smallest integer increment  n = length(anyArray);  i = 2;  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  for i=2:n  x = anyArray(i);  j = i + 1;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '3 ' ];  while ((j > 0) & (anyArray(j) > x)),  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '4 ' ];  anyArray(j+1) = anyArray(j);  j = j - 1;  traversedPath = [traversedPath '3 ' ];  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '5 ' ];  anyArray(j+1) = x;  traversedPath = [traversedPath '2 ' ];  end  sortedArray = anyArray;  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '6 ' ];  end |

Lampiran 7 CFG dan *Cyclomatic Complexity* iA2008



***Cyclomatic Complexity*:**

N(G) = 6

E(G) = 7

V(G) = E - N + 2

= 3

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

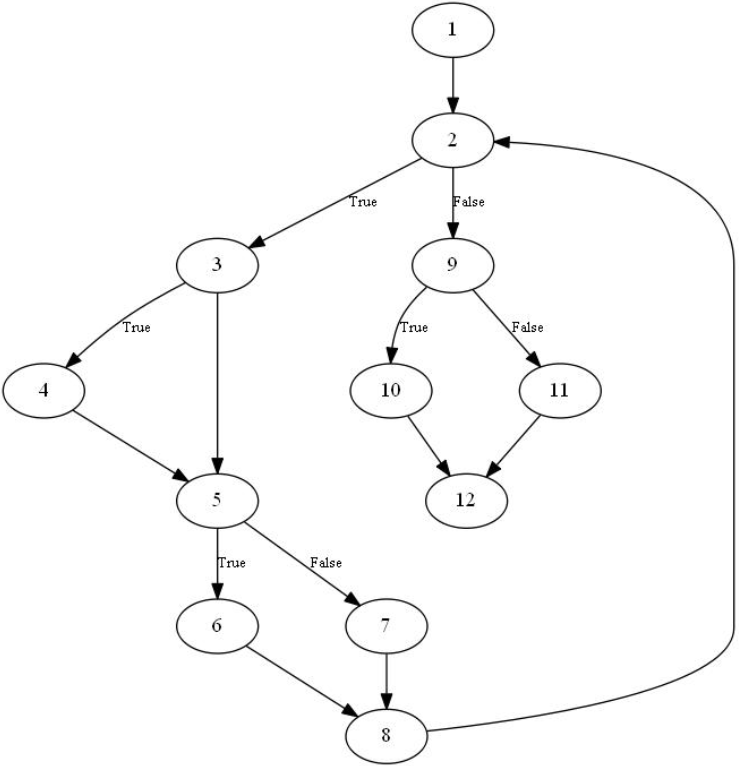
Lampiran 9 Kode program binA2008

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

Lampiran 10 Hasil instrumentasi binA2008

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9**  **Node 10**  **Node 11**  **Node 12** | function [traversedPath,itemIndex] = binary(itemNumbers)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  item = itemNumbers(1);  numbers = itemNumbers(1,2:end);  lowerIdx = 1;  upperIdx = length(numbers);  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  while (lowerIdx ~= upperIdx), % Branch # 1  temp = lowerIdx + upperIdx; % additional statement  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '3 ' ];  if (mod(temp, 2) ~= 0),  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '4 ' ];  temp = temp - 1;  end % additional statement  idx = temp / 2;  % instrument Branch # 3  traversedPath = [traversedPath '5 ' ];  if (numbers(idx) < item), % Branch # 2  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '6 ' ];  lowerIdx = idx + 1;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '7 ' ];  upperIdx = idx;  end  traversedPath = [traversedPath '8 ' ];  traversedPath = [traversedPath '2 ' ];  end  % Additional code that returns -1 if the item is not found  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 4  traversedPath = [traversedPath '9 ' ];  if (item == numbers(lowerIdx)),  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '10 ' ];  temIndex = lowerIdx;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '11 ' ];  itemIndex = -1;  end  traversedPath = [traversedPath '12 ' ];  end |

Lampiran 11 CFG dan *Cyclomatic Complexity* binA2008



***Cyclomatic Complexity*:**

N(G) = 12

E(G) = 15

V(G) = E - N + 2

= 5

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

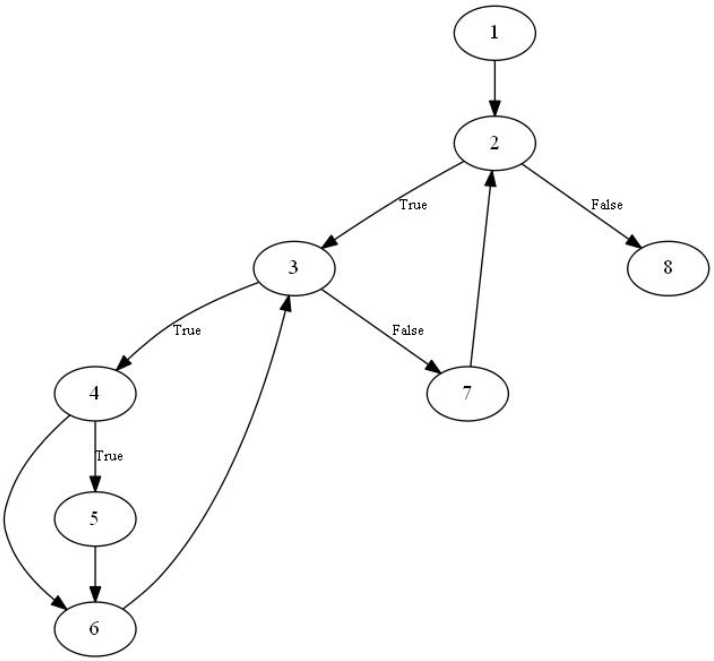
Lampiran 13 Kode program bubA2008

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

Lampiran 14 Hasil instrumentasi bubA2008

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8** | function [traversedPath,sortedArray] = bubble(anyArray)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  sorted = 0; % 0 means false  i = 1; n = length(anyArray);  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  while ((i <= (n-1)) && ~sorted), % Branch # 1  sorted = 1;  j = n;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '3 ' ];  for j=n:-1:i+1 % Branch # 2  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 3  traversedPath = [traversedPath '4 ' ];  if (anyArray(j) < anyArray(j-1)) % Branch # 3  %exchange(anyArray(j), anyArray(j-1));  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '5 ' ];  temp = anyArray(j);  anyArray(j) = anyArray(j-1);  anyArray(j-1) = temp;  sorted = 0;  end  traversedPath = [traversedPath '6 ' ];  traversedPath = [traversedPath '3 ' ];  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '7 ' ];  i = i + 1;  traversedPath = [traversedPath '2 ' ];  end  sortedArray = anyArray;  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '8 ' ];  end |

Lampiran 15 CFG dan *Cyclomatic Complexity* bubA2008



***Cyclomatic Complexity*:**

N(G) = 8

E(G) = 10

V(G) = E - N + 2

= 4

Lampiran 16 Semua kemungkinan jalur bubA2008

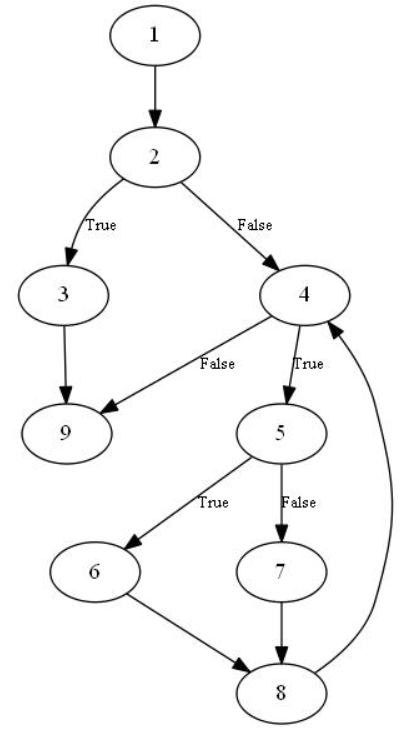
* 1 2 (T) 3 (T) 4 (T) 5 6 3 (F) 7 2 (F) 8
* 1 2 (T) 3 (T) 4 6 3 (F) 7 2 (F) 8
* 1 2 (T) 3 (F) 7 2 (F) 8
* 1 2 (F) 8

Lampiran 17 Kode program gA2008

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

Lampiran 18 Hasil instrumentasi gA2008

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9** | function [traversedPath,y] = gcd(number)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  a = number(1);  b = number(2);  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  if (a == 0),  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '3 ' ];  y = b;  else  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '4 ' ];  while b ~= 0  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 3  traversedPath = [traversedPath '5 ' ];  if a > b  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '6 ' ];  a = a - b;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '7 ' ];  b = b - a;  end  traversedPath = [traversedPath '8 ' ];  traversedPath = [traversedPath '4 ' ];  end  y = a;  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '9 ' ];  end |

Lampiran 19 CFG dan *Cyclomatic Complexity* gA200

***Cyclomatic Complexity*:**

N(G) = 9

E(G) = 11

V(G) = E - N + 2

= 4

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  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9**  **Node 10**  **Node 11**  **Node 12**  **Node 13**  **Node 14**  **Node 15**  **Node 16**  **Node 17**  **Node 18**  **Node 19**  **Node 20**  **Node 21** | function result = expintBueno2002(numbersIn)  n = numbersIn(1); % integer  x = numbersIn(2); % floa  MAXIT = 100;  EULER = 0.5772156649;  FPMIN = 1.0e-30;  EPS = 1.0e-7;  nm1 = n - 1;  if (n < 0 || x < 0.0 || (x == 0.0 && (n == 0.0 || n==1)))  result = 0;  % disp(‘bad arguments in expintBueno2002’);  elseif (n == 0)  result = exp(-x)/x;  elseif (x == 0.0)  result = 1.0/nm1; % strangy: what is nm1?  elseif (x > 1.0)  b = x + n;  c = 1.0 / FPMIN;  d = 1.0 / b;  h = d;  for i=1 : MAXIT  a = -i \* (nm1 + i);  b = b + 2.0;  d = 1.0 / (a\*d+b);  c = b + a / c;  del = c \* d;  h = h \* del;  if (abs(del-1.0) < EPS) % abs is fabs in C  result = h \* exp(-x);  return;  end  end  disp(‘continuated fraction failed in expint’);  else  % ans = (nm1!=0 ? 1.0/nm1 : -log(x)-EULER);  % is interpreted as follows  if (nm1 ~= 0)  result = 1.0 / nm1;  else  result = -log(x)-EULER;  end  fact = 1.0;  for i = 1 : MAXIT  fact = fact \* (-x / i);  if (i ~= nm1)  del = -fact / (i - nm1);  else  psi = -EULER;  for ii = 1 : nm1  psi = psi + (1/ii);  end  del = fact \* (-log(x) + psi);  end  result = result + del;  if (abs(del) < abs(result) \* EPS) % abs is fabs in C  return;  end  end  disp(‘series failed in expint’);  end  end |

Lampiran 22 Hasil instrumentasi eB2002

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9**  **Node 11**  **Node 12**  **Node 13**  **Node 14**  **Node 15**  **Node 16**  **Node 17**  **Node 18**  **Node 19**  **Node 20**  **Node 21**  **Node 10** | function [traversedPath,result] = expintBueno2002(numbersIn)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  n = numbersIn(1); % integer  x = numbersIn(2); % floa  MAXIT = 100;  EULER = 0.5772156649;  FPMIN = 1.0e-30;  EPS = 1.0e-7;  nm1 = n - 1;  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  if (n < 0 || x < 0.0 || (x == 0.0 && (n == 0.0 || n==1)))  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '3 ' ];  result = 0;  % disp(‘bad arguments in expintBueno2002’);  elseif (n == 0)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '4 ' ];  result = exp(-x)/x;  elseif (x == 0.0)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '5 ' ];  result = 1.0/nm1; % strangy: what is nm1?  elseif (x > 1.0)  b = x + n;  c = 1.0 / FPMIN;  d = 1.0 / b;  h = d;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '6 ' ];  for i=1 : MAXIT  a = -i \* (nm1 + i);  b = b + 2.0;  d = 1.0 / (a\*d+b);  c = b + a / c;  del = c \* d;  h = h \* del;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 3  traversedPath = [traversedPath '7 ' ];  if (abs(del-1.0) < EPS) % abs is fabs in C  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '8 ' ];  result = h \* exp(-x);  return;  end  traversedPath = [traversedPath '9 ' ];  traversedPath = [traversedPath '6 ' ];  traversedPath = [traversedPath '14 ' ];  end  disp('continuated fraction failed in expint');  else  % ans = (nm1!=0 ? 1.0/nm1 : -log(x)-EULER);  % is interpreted as follows  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 4  traversedPath = [traversedPath '11 ' ];  if (nm1 ~= 0)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '12 ' ];  result = 1.0 / nm1;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '13 ' ];  result = -log(x)-EULER;  end  fact = 1.0;  % instrument Branch # 5  traversedPath = [traversedPath '14 ' ];  for i = 1 : MAXIT  fact = fact \* (-x / i);  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 6  traversedPath = [traversedPath '15 ' ];  if (i ~= nm1)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '16 ' ];  del = -fact / (i - nm1);  else  psi = -EULER;  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 7  traversedPath = [traversedPath '17 ' ];  for ii = 1 : nm1  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '18 ' ];  psi = psi + (1/ii);  traversedPath = [traversedPath '17 ' ];  end  del = fact \* (-log(x) + psi);  end  result = result + del;  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 8  traversedPath = [traversedPath '19 ' ];  if (abs(del) < abs(result) \* EPS) % abs is fabs in C  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '20 ' ];  return;  end  traversedPath = [traversedPath '21 ' ];  end  disp('series failed in expint');  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '10 ' ];  end |

Lampiran 23 CFG dan *Cyclomatic Complexity* eB2002

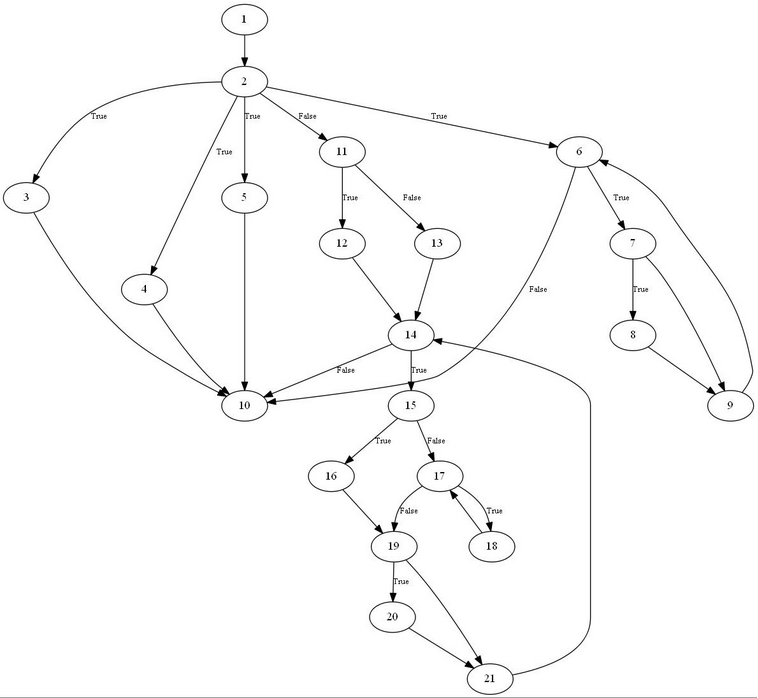
***Cyclomatic Complexity*:**

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

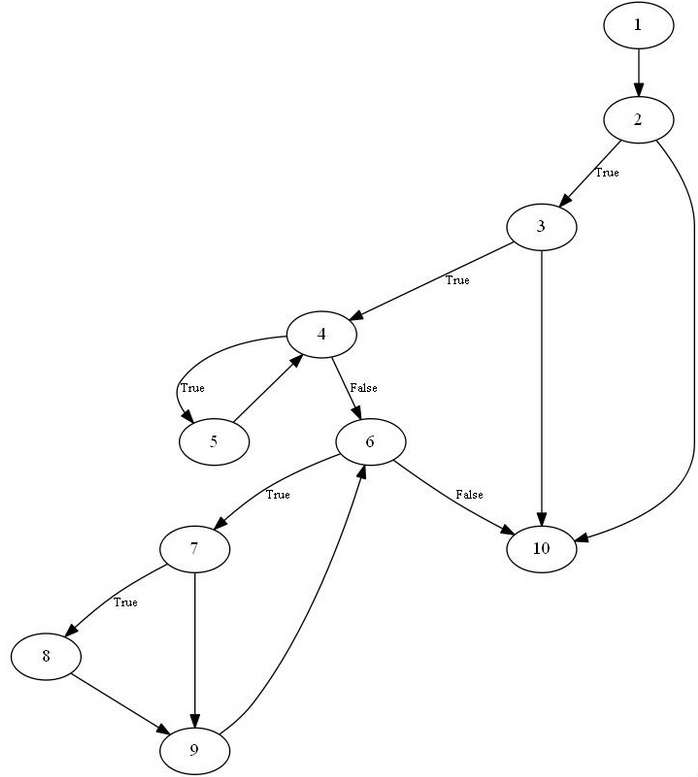
Lampiran 25 Kode program qB2002

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

Lampiran 26 Hasil instrumentasi qB2002

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9**  **Node 10** | function [traversedPath,q, r] = quotientBueno2002(operands)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  n = operands(1); % First number  d = operands(2); % Second number  q = 0;  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  if (d ~= 0)  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '3 ' ];  if ( (d > 0) && (n > 0) )  q = 0;  r = n;  t = d;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 3  traversedPath = [traversedPath '4 ' ];  while (r >= t)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '5 ' ];  t = t \* 2;  traversedPath = [traversedPath '4 ' ];  end  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 4  traversedPath = [traversedPath '6 ' ];  while (t ~= d)  q = q \* 2;  t = t / 2;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 5  traversedPath = [traversedPath '7 ' ];  if (t <= r)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '8 ' ];  r = r - t;  q = q + 1;  end  traversedPath = [traversedPath '9 ' ];  traversedPath = [traversedPath '6 ' ];  end  end  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '10 ' ];  end |

Lampiran 27 CFG dan *Cyclomatic Complexity* qB2002



***Cyclomatic Complexity*:**

N(G) = 10

E(G) = 14

V(G) = E - N + 2

= 6

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

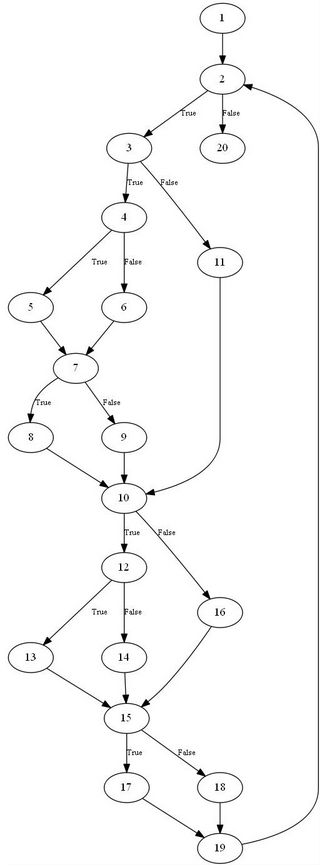
Lampiran 29 Kode program fG2011

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9**  **Node 11**  **Node 10**  **Node 12**  **Node 13**  **Node 14** | function pop = flexGong2011(depop)  [px,py]=size(depop);  for i=1:px % Branch # 1  q=1;  p=depop(i,:);  lex\_compat=p(1);  C\_plus\_plus=p(2);  fulltbl=p(3);  csize =p(4);  unspecified=p(5);  fullspd=p(6);  C\_plus=p(7);  d(1,1)=2;  d(1,1)=1-1.001^(-d(1,1));  d(1,2)=lex\_compat;  d(1,2)=1-1.001^(-d(1,2));  d(1,3)=0;  d(2,1)=2;  d(2,1)=1-1.001^(-d(2,1));  d(2,2)=C\_plus\_plus;  d(2,2)=1-1.001^(-d(2,2));  d(2,3)=0;  d(3,1)=2;  d(3,1)=1-1.001^(-d(3,1));  d(3,2)=fulltbl;  d(3,2)=1-1.001^(-d(3,2));  d(3,3)=0;  d(4,1)=abs( csize-unspecified)+2;  d(4,1)=1-1.001^(-d(4,1));  d(4,2)=2;  d(4,2)=1-1.001^(-d(4,2));  d(4,3)=0;  d(5,1)=2;  d(5,1)=1-1.001^(-d(5,1));  d(5,2)=fullspd;  d(5,2)=1-1.001^(-d(5,2));  d(5,3)=0;  d(6,1)=2;  d(6,1)=1-1.001^(-d(5,1));  d(6,2)=C\_plus;  d(6,2)=1-1.001^(-d(5,2));  d(5,3)=0;  u=3\*ones(1,6);  if (lex\_compat ~= 0) % Branch # 2  d(1,1)=0;  u(1)=1;  if (C\_plus\_plus ~= 0) % Branch # 3  flexerror = 'Can not use -+ with -l option';  d(2,1)=0;  else  d(2,2)=0;  end  if (fulltbl ~= 0) % Branch # 4  flexerror='Can not use -f or -F with -l option';  d(3,1)=0;  else  d(3,2)=0;  end  else  d(1,2)=0;  end  if (csize == unspecified) % Branch # 5  d(4,1)=0;  if (fullspd ~= 0) % Branch # 6  csize = 'DEFAULT\_CSIZE';  d(5,1)=0;  else  d(5,2)=0;  csize = csize;  end |
|  | **Node 16**  **Node 15**  **Node 17**  **Node 18**  **Node 19**  **Node 20** | else  d(4,2)=0;  end  if (C\_plus ~= 0) % Branch # 7  suffix='cc';  d(6,1)=0;  else  d(6,2)=0;  suffix='c';  outfilename = 'outfile\_path';  end  pop(i,:) = p;  end  end |

Lampiran 30 Hasil instrumentasi fG2011

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9**  **Node 11**  **Node 10**  **Node 12**  **Node 13**  **Node 14**  **Node 16**  **Node 15**  **Node 17**  **Node 18**  **Node 19**  **Node 20** | function [traversedPath,pop] = flexGong2011(depop)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  [px,py]=size(depop);  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  for i=1:px % Branch # 1  q=1;  p=depop(i,:);  lex\_compat=p(1);  C\_plus\_plus=p(2);  fulltbl=p(3);  csize =p(4);  unspecified=p(5);  fullspd=p(6);  C\_plus=p(7);  d(1,1)=2;  d(1,1)=1-1.001^(-d(1,1));  d(1,2)=lex\_compat;  d(1,2)=1-1.001^(-d(1,2));  d(1,3)=0;  d(2,1)=2;  d(2,1)=1-1.001^(-d(2,1));  d(2,2)=C\_plus\_plus;  d(2,2)=1-1.001^(-d(2,2));  d(2,3)=0;  d(3,1)=2;  d(3,1)=1-1.001^(-d(3,1));  d(3,2)=fulltbl;  d(3,2)=1-1.001^(-d(3,2));  d(3,3)=0;  d(4,1)=abs( csize-unspecified)+2;  d(4,1)=1-1.001^(-d(4,1));  d(4,2)=2;  d(4,2)=1-1.001^(-d(4,2));  d(4,3)=0;  d(5,1)=2;  d(5,1)=1-1.001^(-d(5,1));  d(5,2)=fullspd;  d(5,2)=1-1.001^(-d(5,2));  d(5,3)=0;  d(6,1)=2;  d(6,1)=1-1.001^(-d(5,1));  d(6,2)=C\_plus;  d(6,2)=1-1.001^(-d(5,2));  d(5,3)=0;  u=3\*ones(1,6);  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '3 ' ];  if (lex\_compat ~= 0) % Branch # 2  d(1,1)=0;  u(1)=1;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 3  traversedPath = [traversedPath '4 ' ];  if (C\_plus\_plus ~= 0) % Branch # 3  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '5 ' ];  flexerror = 'Can not use -+ with -l option';  d(2,1)=0;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '6 ' ];  d(2,2)=0;  end  % instrument Branch # 4  traversedPath = [traversedPath '7 ' ];  if (fulltbl ~= 0) % Branch # 4  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '8 ' ];  flexerror='Can not use -f or -F with -l option';  d(3,1)=0;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '9 ' ];  d(3,2)=0;  end  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '11 ' ];  d(1,2)=0;  end  % instrument Branch # 5  traversedPath = [traversedPath '10 ' ];  if (csize == unspecified) % Branch # 5  d(4,1)=0;  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 6  traversedPath = [traversedPath '12 ' ];  if (fullspd ~= 0) % Branch # 6  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '13 ' ];  csize = 'DEFAULT\_CSIZE';  d(5,1)=0;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '14 ' ];  d(5,2)=0;  csize = csize;  end  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '16 ' ];  d(4,2)=0;  end  % instrument Branch # 7  traversedPath = [traversedPath '15 ' ];  if (C\_plus ~= 0) % Branch # 7  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '17 ' ];  suffix='cc';  d(6,1)=0;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '18 ' ];  d(6,2)=0;  suffix='c';  outfilename = 'outfile\_path';  end  traversedPath = [traversedPath '19 ' ];  pop(i,:) = p;  traversedPath = [traversedPath '2 ' ];  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '20 ' ];  end |

Lampiran 31 CFG dan *Cyclomatic Complexity* fG2011



***Cyclomatic Complexity*:**

N(G) = 20  
E(G) = 26  
V(G) = E - N + 2  
     = 8

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

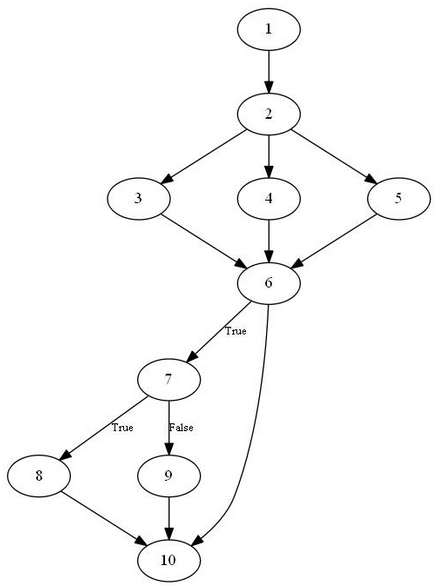
Lampiran 33 Kode program fmH2014

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

Lampiran 34 Hasil instrumentasi fmH2014

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8**  **Node 9**  **Node 10** | function [traversedPath,branchVal] = fitnessMiniMaxi(branchNo, predicate)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  k = 1; % the smallest step for integer  traversedPath = [traversedPath '2 ' ];  switch (branchNo)  case 1,  traversedPath = [traversedPath '3 ' ];  % branch #1: (idx <= numLength)  branchVal = predicate(1) - predicate(2);  case 2,  traversedPath = [traversedPath '4 ' ];  % branch #2: (maxi < num(idx))  branchVal = predicate(1) - predicate(2);  case 3,  traversedPath = [traversedPath '5 ' ];  % branch #3: (mini > num(idx))  branchVal = predicate(2) - predicate(1);  end  % instrument Branch # 1  traversedPath = [traversedPath '6 ' ];  if ((branchNo == 2) || (branchNo == 3)),  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '7 ' ];  if (branchVal < 0)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '8 ' ];  branchVal = branchVal - k;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '9 ' ];  branchVal = branchVal + k;  end  end  traversedPath = [traversedPath '10 ' ];  end |

Lampiran 35 CFG dan *Cyclomatic Complexity* fmH2014



***Cyclomatic Complexity*:**

N(G) = 10  
E(G) = 13  
V(G) = E - N + 2  
     = 5

Lampiran 36 Semua kemungkinan jalur fmH2014

* 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.