

**ILMU KOMPUTER**

**FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM**

**INSTITUT PERTANIAN BOGOR**

**BOGOR**

**2017**

**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 Fermentasi 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, Desember 2017

*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 dan bertujuan untuk menemukan kesalahan pada perangkat lunak. Path testing merupakan salah satu metode pengujian struktural yang menggunakan source code dari program untuk menemukan semua jalur yang mungkin dapat dilalui program dan memastikan semua jalur dijalankan setidaknya satu kali. Untuk menguji perangkat lunak yang kompleks secara keseluruhan akan memakan waktu yang lama dan membutuhkan sumber daya manusia yang banyak. Pada penelitian ini, akan dibangun sebuah sistem 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 memastikan semua jalur dijalankan setidaknya 1 kali ketika diberikan masukan data uji, maka sistem ini juga akan melakukan instrumentasi kode program secara otomatis. Program yang akan diuji dalam penelitian ini adalah program yang dibangun dengan menggunakan bahasan C. Dalam pengembangannya, aplikasi ini akan dibangun dengan menggunakan bahasa pemrogramana C# dan library Graphvis untuk memvisualisasikan Control Flow Graph.

Kata kunci: Path Testing; Control Flow Graph, Instrumentasi

**ABSTRACT**

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

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**

**2017**

**INSTRUMENTASI *SOURCE CODE* 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

Dr Ir Agus Buono, MSi MKom

Ketua Departemen

Tanggal Lulus: (tanggal penandatanganan skripsi oleh ketua departemen)

**PRAKATA**

Bogor, Januari 2018

*Raden Asri Ramadhina Fitriani*

**DAFTAR ISI**

DAFTAR TABEL vi

DAFTAR GAMBAR vi

DAFTAR LAMPIRAN vi

[PENDAHULUAN 1](#_Toc505072328)

[Latar Belakang 1](#_Toc505072329)

[Perumusan Masalah 1](#_Toc505072330)

[Tujuan Penelitian 2](#_Toc505072331)

[Manfaat Penelitian 2](#_Toc505072332)

[Ruang Lingkup Penelitian 2](#_Toc505072333)

[METODE 2](#_Toc505072334)

[Analisis 2](#_Toc505072335)

[Perancangan 3](#_Toc505072336)

[Implementasi 6](#_Toc505072337)

[Testing 6](#_Toc505072338)

[HASIL DAN PEMBAHASAN 7](#_Toc505072339)

[Analisis 7](#_Toc505072340)

[Perancangan 8](#_Toc505072341)

[Implementasi 10](#_Toc505072342)

[Testing 20](#_Toc505072343)

[SIMPULAN DAN SARAN 21](#_Toc505072344)

[Simpulan 21](#_Toc505072345)

[Saran 21](#_Toc505072346)

[DAFTAR PUSTAKA 21](#_Toc505072347)

LAMPIRAN 13

RIWAYAT HIDUP 15

**DAFTAR TABEL**

[Contoh program uji 7](#_Toc505072348)

**DAFTAR GAMBAR**

[Tahapan penelitian 2](#_Toc505072349)

[Arsitektur sistem 3](#_Toc505072350)

[Contoh hasil penguraian ke dalam dokumen XML menggunakan library MATLAB-PARSER 4](#_Toc505072351)

[Notasi Control Flow Graph (CFG) 5](#_Toc505072352)

[Contoh bahasa dot 5](#_Toc505072353)

[Contoh hasil visualisasi bahasa dot menggunakan library graphviz 5](#_Toc505072354)

[Perancangan *Class Diagram* 8](#_Toc505072355)

[Perancangan antarmuka *form awal* 9](#_Toc505072356)

[Perancangan antarmuka hasil dari proses yang telah dilakukan 10](#_Toc505072357)

[Potongan hasil penguraian kode program tA2008 ke dalam format XML 12](#_Toc505072358)

[Hasil *nodes* yang terbentuk dari tA2008 13](#_Toc505072359)

[*Object diagram* tA2008 14](#_Toc505072360)

[Representasi tA2008 dalam bahasa dot 15](#_Toc505072361)

[CFG tA2008 16](#_Toc505072362)

[Hasil instrumentasi tA2008 17](#_Toc505072363)

[Hasil ekseskusi kode program tA2008 yang sudah diinstrumentasi 17](#_Toc505072364)

[Tampilan halaman awal apikasi 18](#_Toc505072365)

[Tampilan hasil pembangkitan 19](#_Toc505072366)

**DAFTAR LAMPIRAN**

[mmA2008 23](#_Toc505072367)

[Kode program iA2008 25](#_Toc505072368)

[Hasil instrumentasi iA2008 25](#_Toc505072369)

[binA2008 26](#_Toc505072370)

[bubA2008 30](#_Toc505072371)

[gA2008 31](#_Toc505072372)

[eB2002 1.47 s 34](#_Toc505072373)

[qB2002 39](#_Toc505072374)

[fB2002 43](#_Toc505072375)

[fmH2014 47](#_Toc505072376)

mmA2008 23

iA2008 25

binA2008 26

bubA2008 30

gA2008 31

eB2002 34

qB2002 39

fB2002 43

fmH2014 47

# 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 source code dari 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 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 untuk membangkitkan data uji untuk path testing menggunakan algoritma genetika. Dalam penelitian tersebut, Hermadi membangkitkan Control flow Graph (CFG) dan instrumentasi masih secara manual sehingga membutuhkan banyak waktu dan rawan akan kesalahan ketika program  
sudah semakin besar. Sehingga mengotomasi hal tersebut dapat membuat path testing menjadi lebih cepat dan 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 re-engineering perangkat lunak.

## Tujuan Penelitian

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

## Manfaat Penelitian

Hasil penelitian diharapkan dapat membantu pengembang dan penguji aplikasi untuk:

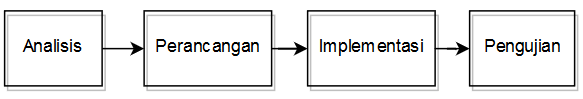
1. Menyisipkan tag-tag sebagai instrumentasi program ke dalam source code secara otomatis sehingga proses tersebut dapat dilakukan dengan lebih cepat.
2. Membangkitkan jalur-jalur dasar yang dapat digunakan sebagai dasar untuk pembangkitan data uji.
3. Membangkitkan diagram CFG yang dapat memudahkan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

## Ruang Lingkup Penelitian

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

# METODE

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



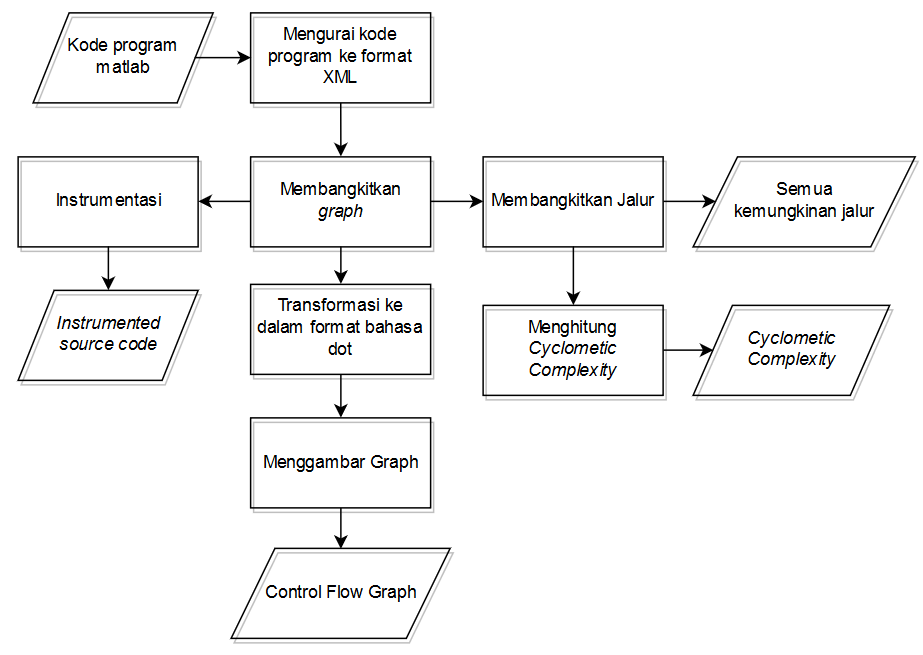
Gambar 1 Tahapan penelitian

## **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 digunakan dalam 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 matlabakan dibaca sebagai inputan. Matlab memiliki empat struktur kontrol, yaitu IF-ELSE-END, SWITCH-CASE, FOR, dan WHILE. Kode program yang diinputkan harus sudah dipastikan dapat dijalankan jika di *compile.*

**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 dibuat 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). Contoh hasil penguraian dari struktur control perintah IF-ELSE-END menggunakan *library* MATLAB-PARSER dapat dilihat pada Gambar 3

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **🡪 node 1**  **🡪 node 4**  **🡪 node 3**  **🡪 node 2**  <If Line="5" Column="2" Text="if">  <If.IfPart>  <IfPart Line="5" Column="2" Text="if">  <IfPart.Expression>  </IfPart.Expression>  <IfPart.Statements>  </IfPart.Statements>  </IfPart>  </If.IfPart>  <If.ElsePart>  <ElsePart Line="30" Column="2" Text="else">  <ElsePart.Statements>  </ElsePart.Statements>  </ElsePart>  </If.ElsePart>  <If.Terminator>  </If.Terminator>  </If> |

Gambar 3 Contoh hasil penguraian ke dalam dokumen XML menggunakan library MATLAB-PARSER

**Membangkitkan *Graph***

Setiap dalam tag 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 *nodes* baru di dalam kelas *graph*. Setiap *node* berisi informasi nomor baris dan 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:

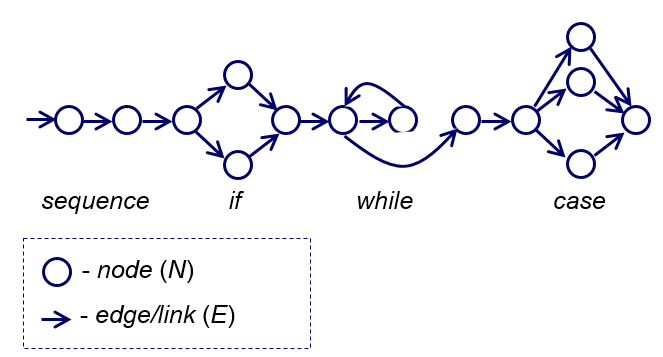
1. Menggambarkan jalur dalam bentuk *Control Flow Graph* (CFG)
2. Menghitung *cyclomatic complexity*
3. Memilih satu set jalur dasar
4. Membangkitkan data uji untuk setiap jalur dasar

**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).

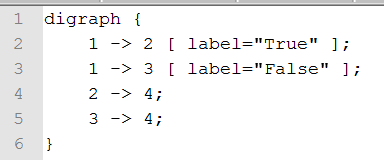
**Memvisualisasikan *Graph* dalam bentuk CFG**

*Control Flow Graph* (CFG) adalah graph berarah yang merepresentasikan aliran dari sebuah program. Setiap CFG terdiri dari *nodes* dan *edges*. *Nodes* merepresentasikan *statement* atau *expressions*. Sedangkan *edges* merepresentasikan transfer kontrol antar *nodes* (Watson dan McCabe 1996). Notasi dari CFG dapat dilihat pada Gambar 4.

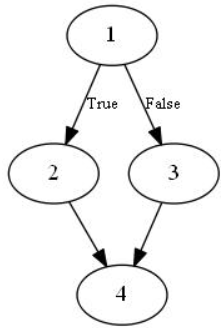


Gambar 4 Notasi Control Flow Graph (CFG)

Setelah file dengan format bahasa dot terbentuk, CFG akan divisualisasikan dengan menggunakan library Graphviz. Graphviz merupakan perangkat lunak *open source* untuk visualisasi grafik. Gambar 5 menunjukkan contoh bahasa dot yang terdiri dari 4 *nodes* dan 4 *edges.* Gambar 6 menunjukkan hasil dari *graph*  yang dibentuk dari perintah dot pada Gambar 5 yang di visualisasikan menggunakan Graphviz.



Gambar 5 Contoh bahasa dot



Gambar 6 Contoh hasil visualisasi bahasa dot menggunakan library graphviz

**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 dilakukan dengan cara menambahkan dulu *variable* keluaran bernama *traversedPath*. *Variable* 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. *Library* yang dibutuhkan adalah *library* MATLAB-PARSER yang dibuat oleh Suffos (2015) untuk mengurai kode program ke dalam bentuk XML.

Setelah file dengan format bahasa dot terbentuk, CFG divisualisasikan dengan menggunakan *library* Graphviz.Net*.* Graphviz.Net adalah pembungkus C# untuk generator grafik Graphviz yang dibuat 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 dilakukan dengan membandingkan hasil yang dikeluarkan oleh sistem dengan pembangkitan secara manual dari segi waktu eksekusi.

# 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.

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.

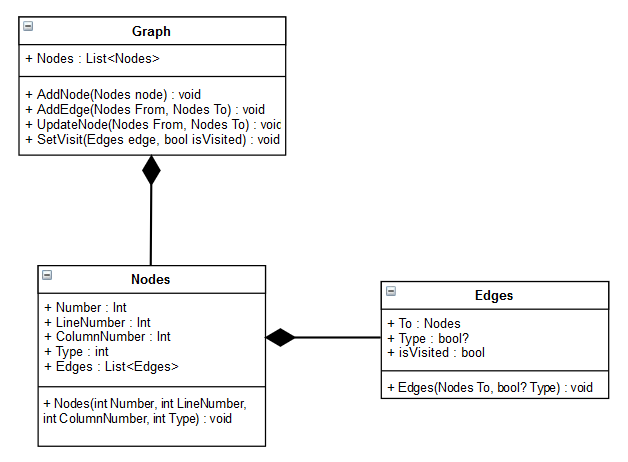
Tabel 1 Contoh program uji

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Program Uji** | **Nama** | **Deskripsi** |
| 1 | triangleAhmed2008 | tA2008 | Menentukan tipe dari segitiga apakah termasuk *equilateral, isosceles, scalene,* atau *not triangle* |
| 2 | minimaxiAhmed2008 | mmA2008 | Menentukan nilai minimal dan maksimal dari inputan berupa bilangan dalam *array* |
| 3 | insertionAhmed2008 | iA2008 | Mengurutkan bilangan dalam *array* menggunakan metode *insertion sort* |
| 4 | binnaryAhmed2008 | binA2008 | mencari indeks sebuah bilangan dalam array dengan mengembalikan indeks jika ditemukan dan tidak jika tidak ditemukan. |
| 5 | bubbleAhmed2008 | bubA2008 | Mengurutkan bilangan dalam *array* menggunakan metode *bubble sort* |
| 6 | gcdAhmed2008 | gA2008 | Menghitung *GCD* atau pembagi dua bilangan terbesar |
| 7 | expintBueno2002 | eB2002 | Fungsi exponensial yang dapat memproses bilangan *integer* dan *float* |
| 8 | quotientBueno2002 | qB2002 | Menghitung hasil bagi dan sisa hasil bagi dari dua buah bilangan bulat positif |
| 9 | findBueno2002 | fB2002 | Mengurutkan bilangan dalam *array* sebagian |
| 10 | fitnessMiniMaxiHermadi2014 | fmH2014 | Menghitung fungsi *fitness* dari fungsi minimaxiAhmed2008 |

## **Perancangan**

**Perancangan *Class Diagram***

*Class diagram* dibangun untuk menggambarkan struktur sistem dari segi pendefinisian *class-class*. Perancangan *Class diagram* dapat dilihat pada Gambar 7.

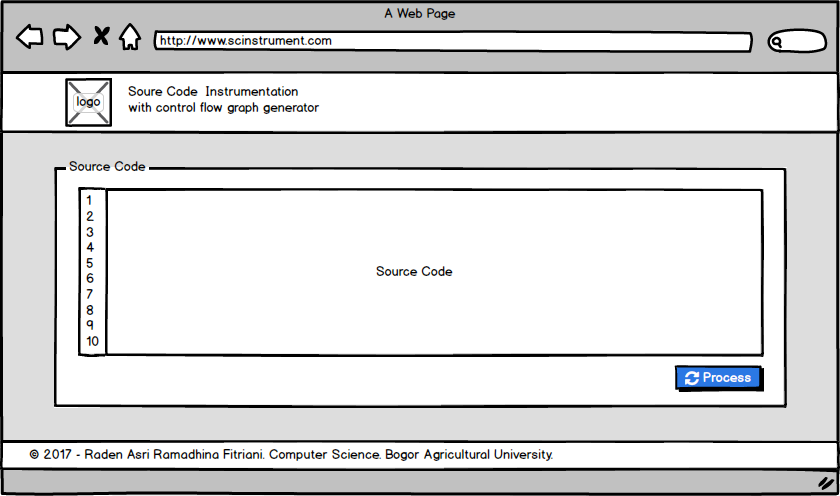


Gambar 7 Perancangan *Class Diagram*

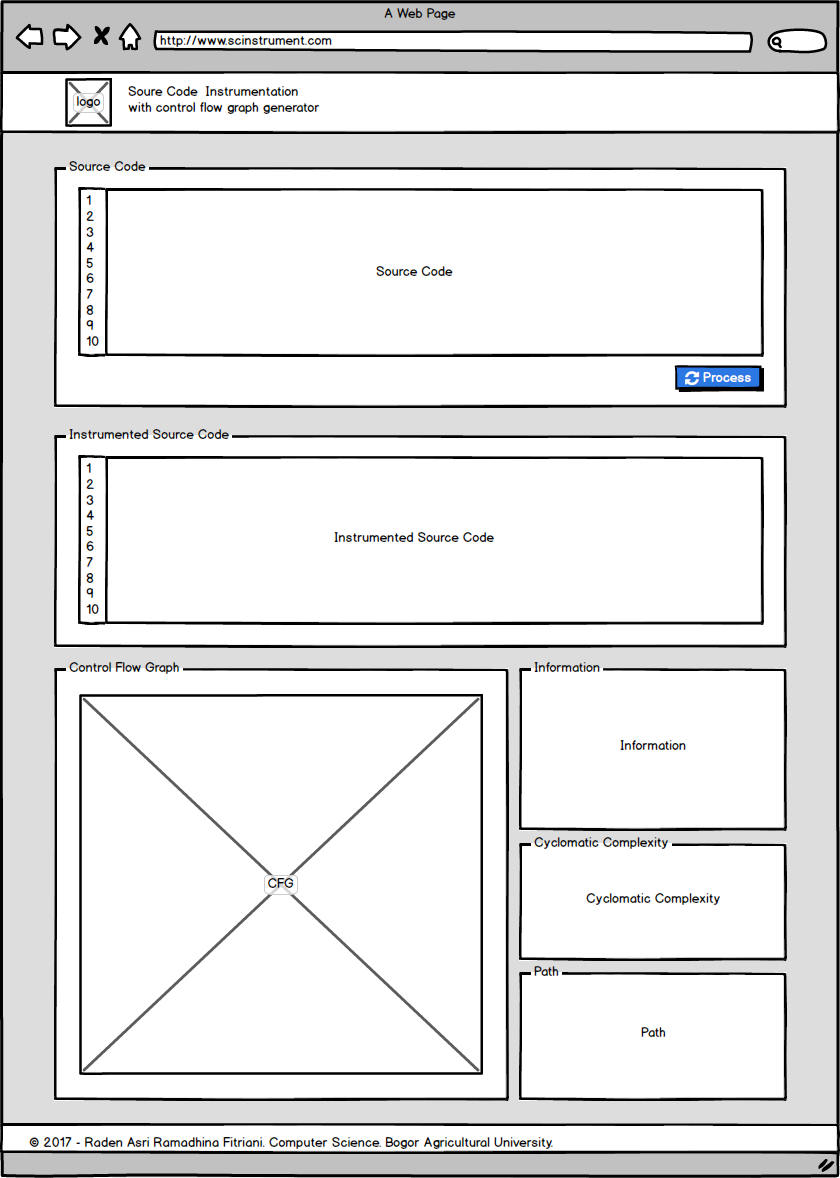
*Class graph* memiliki atribut berupa list dari *class node.* Dalam sebuah *class node* terdapat informasi nomor *node*, nomor baris dan kolom dari kode program, tipe dari perintah tersebut apakah termasuk percabangan, pengulangan, atau berupa perintah biasa. Selain itu, terdapat list *edge* yang berisi *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. Perancangan antarmuka *form* awal yang digunakan untuk memasukkan kode program yang akan di proses dapat dilihat pada Gambar 8 dan perancangan antarmuka hasil dari proses yang telah dilakukan dapat dilihat pada Gambar 9



Gambar 8 Perancangan antarmuka *form awal*



Gambar 9 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 | 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 |

**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 10 menunjukkan potongan hasil penguraian kode program ke dalam format XML. Potongan kode XML yang terlihat pada Gambar 10 menunjukkan hasil penguraian dari kode program pada baris ke 6 sampai baris ke 7.



Gambar 10 Potongan hasil penguraian kode program tA2008 ke dalam format XML

**Membangkitkan Graph**

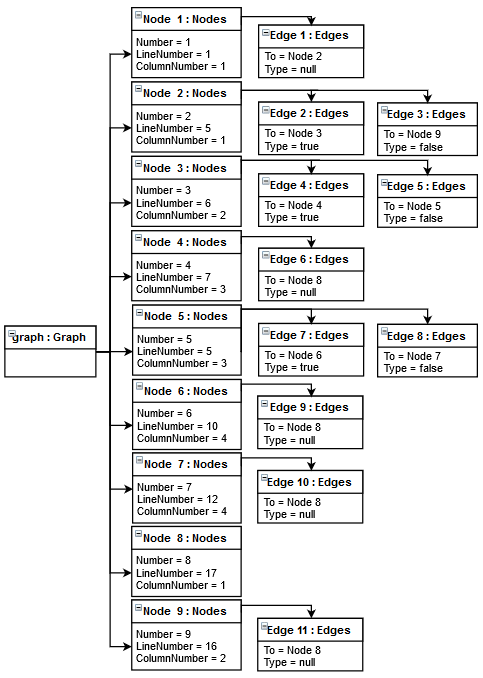
Salah satu cara untuk membaca dan menulis dokumen XML pada *framework* .NET dan C# yaitu dengan menggunakan kelas *XMLDocument* yang terdapat dalam *namespace System.XML*. Setiap elemen XML yang merupakan struktur kontrol pada program akan menjadi *node* baru di dalam kelas *graph*. Setiap *node* berisi informasi nomor baris dan 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.

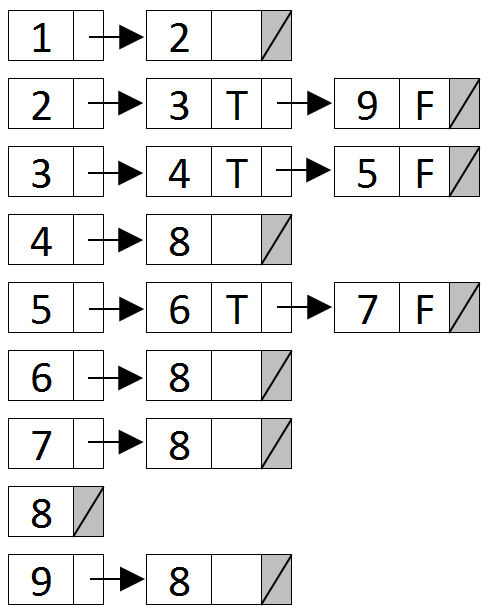
Hasil *node* yang dibentuk dari kode program tA2008 dapat dilihat pada Gambar 11. *Node* 1 dibentuk pada awal kode program sebagai inisialisasi. *Node* ditambahkan ketika bertemu dengan perintah yang termasuk ke dalam struktur kontrol seperti IF-ELSE-END, SWITCH-CASE, FOR, dan WHILE. Seperti dapat dilihat 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 11 Hasil *nodes* yang terbentuk dari tA2008

Representasi objek dari kelas *graph* yang terbentuk dari kode program tA2008 dapat dilihat pada Gambar 12. Terbentuk 9 buah *nodes* dan 11 buah *edges* yang menghubungkan antar *nodes* tersebut.





Gambar 12 *Object diagram* tA2008

**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**

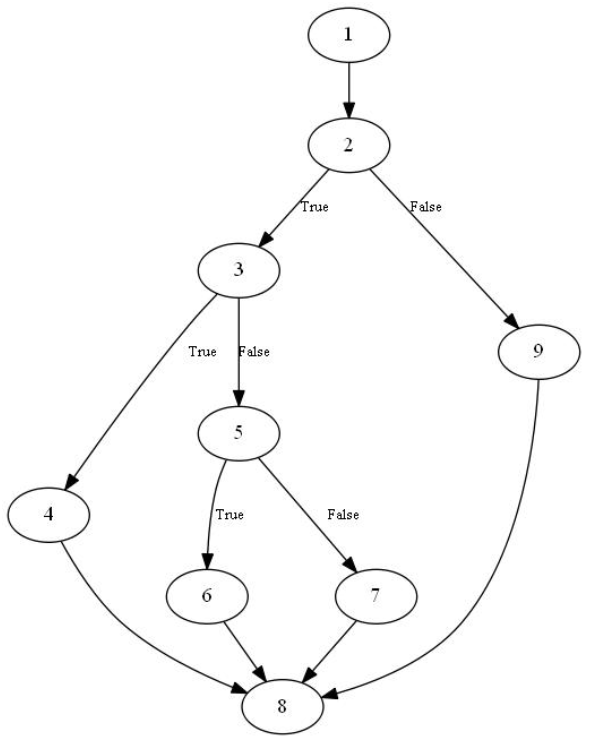
Transformasi ke dalam format bahasa dot dilakukan dengan cara menelusuri objek *graph* yang sudah dibangun sebelumnya. Yang didefinisikan dalam bahasa dot adalah *edge* yang terdapat pada *graph* yang dibangun. Seperti yang dapat dilihat pada Gambar 13, jumlah baris sebanyak jumlah *edge* pada objek *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 13 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 14.



Gambar 14 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 12 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**

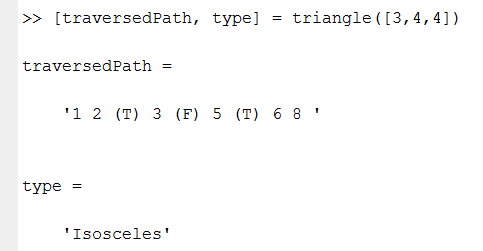
Instrumentasi dilakukan dengan cara menambahkan dulu variabelkeluaran bernama *traversedPath*. Variabelin digunakan untuk menyimpan informasi *node* mana saja yang dilalui ketika diberikan inputan dengan nilai tertentu. Dan menyimpan informasi pilihan yang dilalui ketika ditemukan cabang yang terdapat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

Hasil kode program yang telah diinstrumentasi dapat dilihat pada Gambar 15. 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 16.

Sehingga ketika kode program hasil instrumentasi dijalankan dengan inputan tertentu akan menghasilkan keluaran variabel *traversedPath* dan *type* seperti yang dapat dilihat pada Gambar 16. 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 |

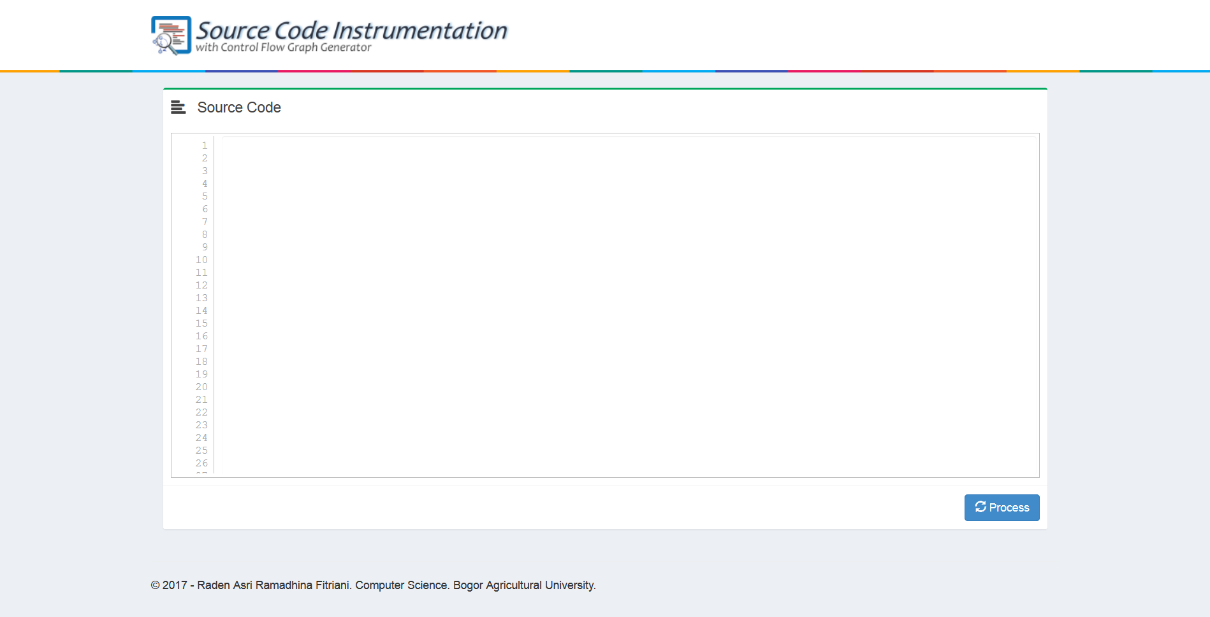
Gambar 15 Hasil instrumentasi tA2008



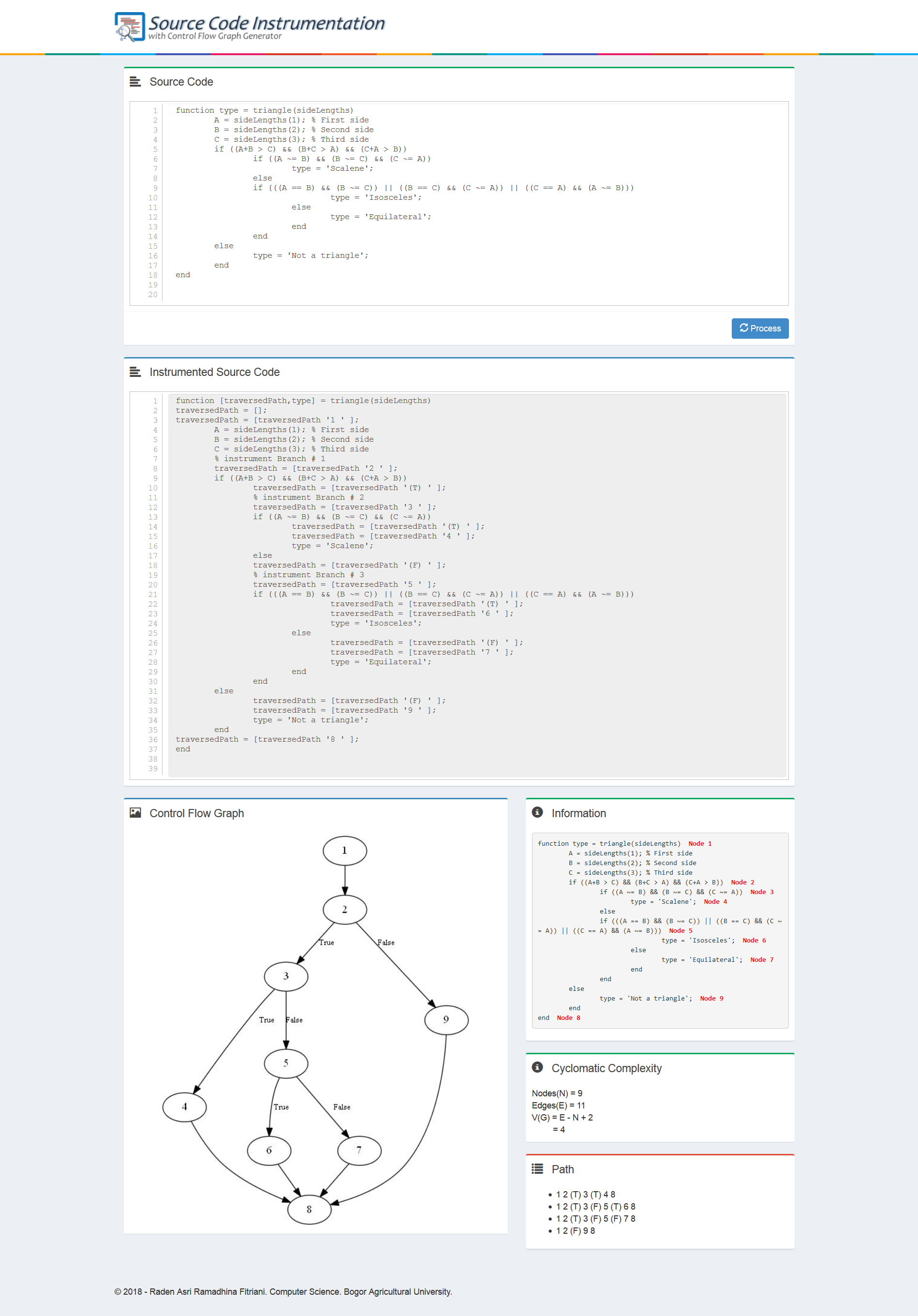
Gambar 16 Hasil ekseskusi kode program tA2008 yang sudah diinstrumentasi

**Implementasi Antarmuka**

Halaman awal dari aplikasi dapat dilihat pada Gambar 17 dimana hanya terdapat satu text area untuk memasukkan kode program yang akan di proses. Tampilan hasil pembangkitan dapat dilihat pada Gambar 18.



Gambar 17 Tampilan halaman awal apikasi



Gambar 18 Tampilan hasil pembangkitan

## Testing

Tahapan ini adalah melakukan evaluasi dari tahapan implementasi. Evaluasi dilakukan dengan membandingkan hasil yang dikeluarkan oleh sistem dengan pembangkitan secara manual dari segi waktu eksekusi. Penguji terdiri dari dua orang yang berprofesi sebagai pengembang sistem. Hasil jalur yang dibentuk dan nilai *cyclomatic complexity* dari program uji dapat dilihat pada Tabel 2. Hasil perbandingan waktu yang dibutuhkan untuk melakukan pembangkitan secara manual dan oleh aplikasi dapat dilihat pada Tabel 3.

Tabel 2 Hasil jalur dan *cyclomatic complexity* dari program uji

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No** | **Nama Program** | **Jumlah Node** | **Jumlah Edge** | **Cyclomatic Complexity** | **Jumlah jalur yang terbentuk** |
|
| 1 | tA2008 | 9 | 11 | 4 | 4 |
| 2 | mmA2008 | 8 | 10 | 4 | 5 |
| 3 | iA2008 | 6 | 7 | 3 | 3 |
| 4 | binA2008 | 12 | 15 | 5 | 10 |
| 5 | bubA2008 | 8 | 10 | 4 | 4 |
| 6 | gA2008 | 9 | 11 | 4 | 4 |
| 7 | eB2002 | 15 | 20 | 7 | 21 |
| 8 | qB2002 | 10 | 14 | 6 | 8 |
| 9 | fB2002 | 22 | 29 | 9 | 12 |
| 10 | fmH2014 | 10 | 13 | 5 | 9 |

Jumlah jalur yang terbentuk bisa lebih besar dari *cyclomatic complexity* karena jumlah jalur yang tebentuk adalah dari semua kemungkinan. Sedangkan jumlah *cyclomatic complexity* menunjukkan *independent path* yaitu jalur yang setidaknya ada 1 jalur baru yang dilalui.

Tabel 3 Perbandingan eksekusi secara menual dan oleh aplikasi

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **No** | **Nama Program** | **Waktu Eksekusi Oleh Aplikasi** | **Waktu Ekseskusi Manual** | | | **Kesesuaian Hasil** | |
| **Penguji**  **1** | **Penguji**  **2** | **Rata-rata** | **Penguji**  **1** | **Penguji**  **2** |
| 1 | tA2008 | 1.53 s | 492 s |  |  |  |  |
| 2 | mmA2008 | 1.71 s | 9.30 |  |  |  |  |
| 3 | iA2008 | 1.56 s | 3.54 |  |  |  |  |
| 4 | binA2008 | 1.45 s | 6.24 |  |  |  |  |
| 5 | bubA2008 | 1.39 s | 7.28 |  |  |  |  |
| 6 | gA2008 | 2.09 s | 8.12 |  |  |  |  |
| 7 | eB2002 | 1.47 s | 11.27 |  |  |  |  |
| 8 | qB2002 | 1.75 s | 3.42 |  |  |  |  |
| 9 | fB2002 | 1.64 s | 4.36 |  |  |  |  |
| 10 | fmH2014 | 1.81 s | 3.21 |  |  |  |  |

# SIMPULAN DAN SARAN

## Simpulan

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

Hasil penelitian menunjukkan bahwa kecepatan eksekusi xxx% jauh lebih cepat dibandingkan dilakukan secara manual. Selain itu, dari 10 kode program yang digunakan sebagai program uji, terdapat 2 yang hasil jalur yang dilalui kurang dari seharusnya.

## 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 nilai dari fungsi *fitness*.

# DAFTAR PUSTAKA

Bente AD, Rico-Hesse R. 2006. Model of dengue virus infection. *Drug Discov Today Dis Models*. 3(1):97-103. doi: 10.1016/j.ddmod. 2006.03.014.

Bernardo L, Izquierdo A, Prado I, Rosario D, Alvarez M, Santana E, Castro J, Martinez J, Rodriguez R, Morier L *et al*. 2008. Primary and secondary infections of *Macaca fascicularis* monkey with Asian and American genotypes of dengue virus 2. *Clin Vaccine Immunol*. 15(3): 439-446. doi: 10.1128/CVI.00208-07.

Kochel TJ, Watts DM, Gonzalo AS, Ewing DF, Porter KR, Russell KL. 2005. Cross-serotype neutralization of dengue virus in *Aotus nancyme* monkeys. *J Infect Dis*. 191(6):1000-1004. doi:10.1086/427511.

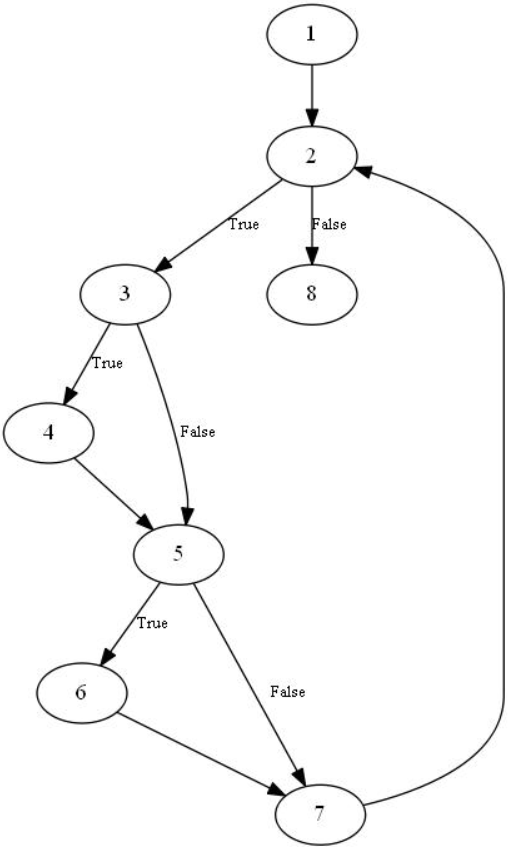
Onlamoon N, Noisakran S, Hsiao HM, Duncan A, Villinger F, Ansari AA, Perng GC. 2010. Dengue virus-induced hemorrhage in a nonhuman primate model. *Blood*. 115(9):1823-1834. doi:10.1182/blood-2009-09-241990.

[WHO] World Health Organization. 2009. Dengue and dengue haemorrhagic fever [internet]. [diacu 2009 Mei 6]. Tersedia dari: http://www.who.int /mediacentre/ factsheets/ fs117/en/ index.html.

Lampiran 1 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 |

|  |  |  |
| --- | --- | --- |
| 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 |



### Cyclomatic Complexity

Nodes(N) = 8  
EdgesI = 10  
V(G) = E – N + 2  
         = 4

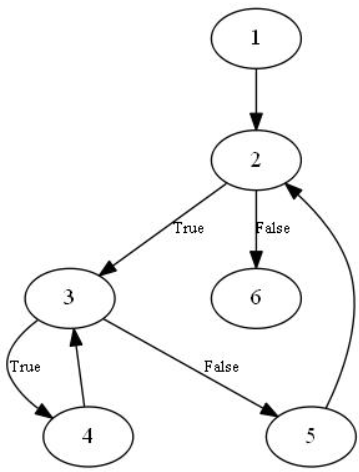
* 1 2 (T) 3 (T) 4 5 (T) 6 7 2 (F) 8
* 1 2 (T) 3 (T) 4 5 (F) 7 2 (F) 8
* 1 2 (T) 3 (F) 5 (T) 6 7 2 (F) 8
* 1 2 (T) 3 (F) 5 (F) 7 2 (F) 8
* 1 2 (F) 8

Lampiran 2 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 3 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)),  anyArray(j+1) = anyArray(j);  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '4 ' ];  j = j - 1;  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '5 ' ];  anyArray(j+1) = x;  end  sortedArray = anyArray;  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '6 ' ];  end |



Nodes(N) = 6  
Edges(E) = 7  
V(G) = E - N + 2  
         = 3

* 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 4 binA2008

function itemIndex = binary(itemNumbers) **Node 1**

item = itemNumbers(1);

numbers = itemNumbers(1,2:end);

lowerIdx = 1;

upperIdx = length(numbers);

while (lowerIdx ~= upperIdx), % Branch # 1 **Node 2**

temp = lowerIdx + upperIdx; % additional statement

if (mod(temp, 2) ~= 0), **Node 3**

temp = temp - 1; **Node 4**

end % additional statement

idx = temp / 2;

if (numbers(idx) < item), % Branch # 2 **Node 5**

lowerIdx = idx + 1; **Node 6**

else

upperIdx = idx; **Node 7**

end **Node 8**

end

% Additional code that returns -1 if the item is not found

if (item == numbers(lowerIdx)), **Node 9**

temIndex = lowerIdx; **Node 10**

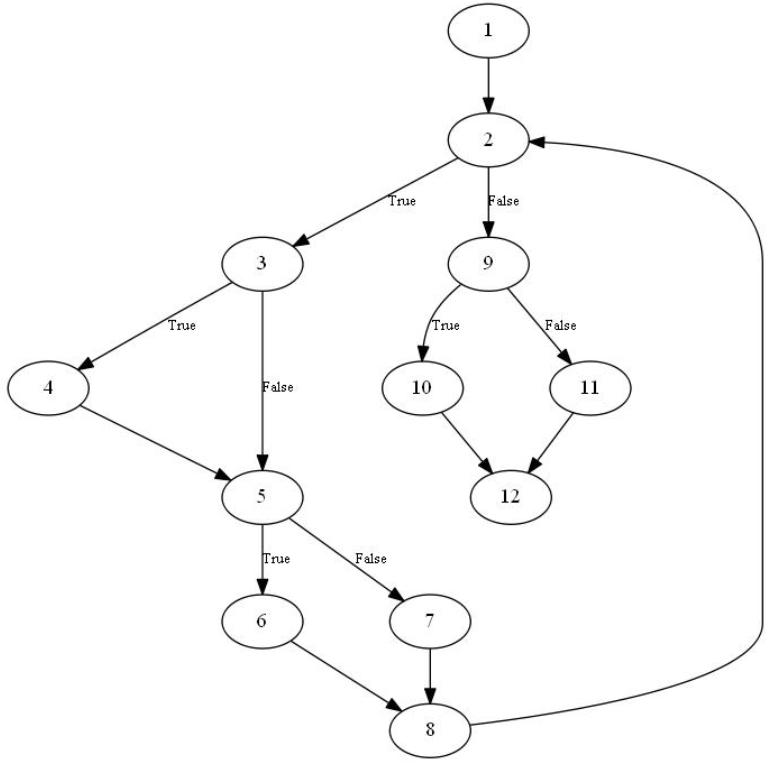
else

itemIndex = -1; **Node 11**

end **Node 12**

end

|  |  |  |
| --- | --- | --- |
| 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 |
| 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,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;  traversedPath = [traversedPath '(F) ' ];  % 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 ' ];  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 |



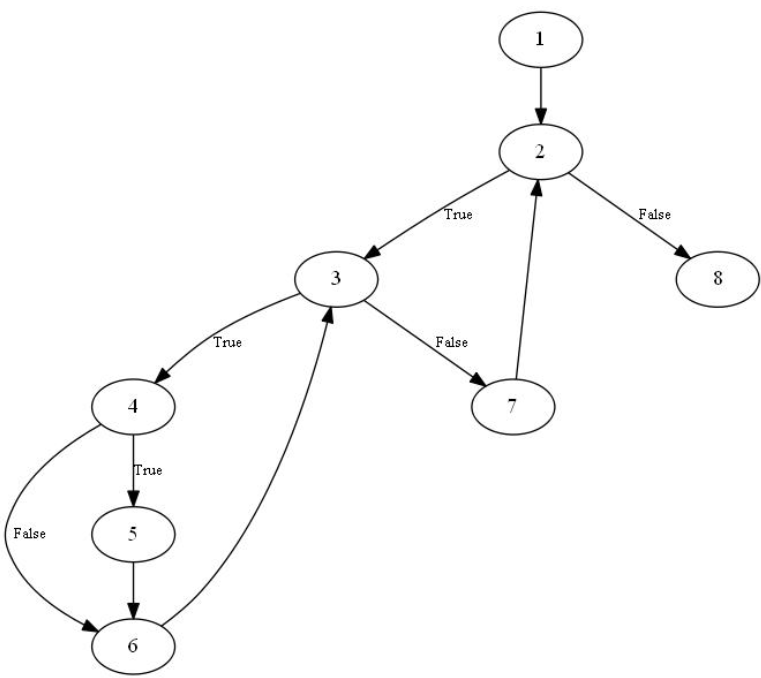
Nodes(N) = 12  
Edges(E) = 15  
V(G) = E - N + 2  
         = 5

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

Lampiran 5 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 |

|  |  |  |
| --- | --- | --- |
| 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 '(F) ' ];  traversedPath = [traversedPath '6 ' ];  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '7 ' ];  i = i + 1;  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '8 ' ];  sortedArray = anyArray;  end |



Nodes(N) = 8  
Edges(E) = 10  
V(G) = E - N + 2  
         = 4

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

Lampiran 6 gA2008

function y = gcd(number) **Node 1**

a = number(1);

b = number(2);

if (a == 0), **Node 2**

y = b; **Node 3**

else

while b ~= 0 **Node 4**

if a > b **Node 5**

a = a - b; **Node 6**

else

b = b - a; **Node 7**

end **Node 8**

end

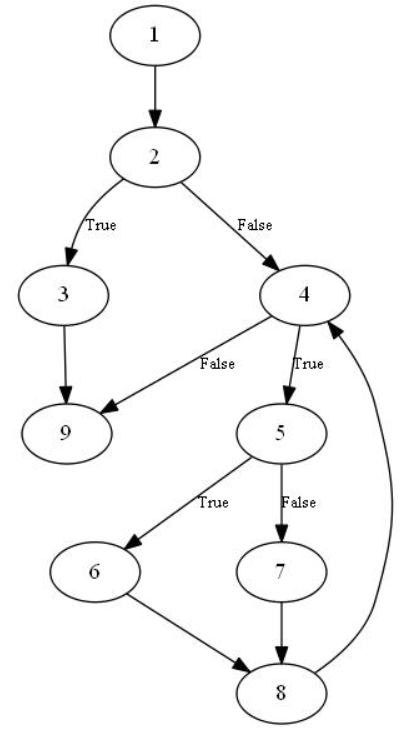
y = a;

end **Node 9**

end

|  |  |  |
| --- | --- | --- |
| 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 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 |

|  |  |  |
| --- | --- | --- |
| 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,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 ' ];  end  y = a;  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '9 ' ];  end |



Nodes(N) = 9  
Edges(E) = 11  
V(G) = E - N + 2  
         = 4

* 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 7 eB2002 1.47 s

function result = expintBueno2002(numbersIn) **Node 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;

if (n < 0 || x < 0.0 || (x == 0.0 && (n == 0.0 || n==1))) **Node 2**

result = 0; **Node 3**

% 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) **Node 4**

result = 1.0 / nm1; **Node 5**

else

result = -log(x)-EULER; **Node 6**

end

fact = 1.0;

for i = 1 : MAXIT **Node 7**

fact = fact \* (-x / i);

if (i ~= nm1) **Node 8**

del = -fact / (i - nm1); **Node 9**

else

psi = -EULER;

for ii = 1 : nm1 **Node 10**

psi = psi + (1/ii); **Node 11**

end

del = fact \* (-log(x) + psi);

end

result = result + del;

if (abs(del) < abs(result) \* EPS) % abs is fabs in C **Node 12**

return;

end **Node 14**

end

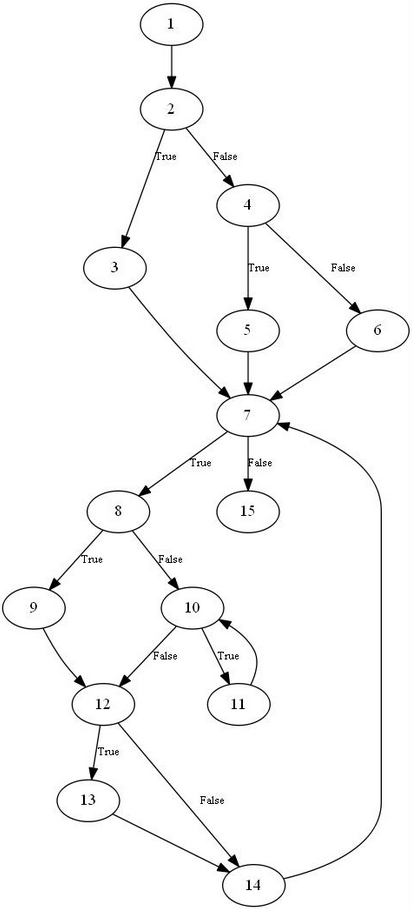
disp('series failed in expint');

end **Node 15**

end

|  |  |  |
| --- | --- | --- |
| 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 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 |

|  |  |  |
| --- | --- | --- |
| 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,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)  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  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '4 ' ];  if (nm1 ~= 0)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '5 ' ];  result = 1.0 / nm1;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '6 ' ];  result = -log(x)-EULER;  end  fact = 1.0;  % instrument Branch # 3  traversedPath = [traversedPath '7 ' ];  for i = 1 : MAXIT  fact = fact \* (-x / i);  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 4  traversedPath = [traversedPath '8 ' ];  if (i ~= nm1)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '9 ' ];  del = -fact / (i - nm1);  else  psi = -EULER;  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 5  traversedPath = [traversedPath '10 ' ];  for ii = 1 : nm1  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '11 ' ];  psi = psi + (1/ii);  end  del = fact \* (-log(x) + psi);  end  result = result + del;  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 6  traversedPath = [traversedPath '12 ' ];  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '13 ' ];  if (abs(del) < abs(result) \* EPS) % abs is fabs in C  return;  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '14 ' ];  end  disp('series failed in expint');  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '15 ' ];  end |



Nodes(N) = 15  
Edges(E) = 20  
V(G) = E - N + 2  
         = 7

### Path

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

Lampiran 8 qB2002

function [q, r] = quotientBueno2002(operands) **Node 1**

n = operands(1); % First number

d = operands(2); % Second number

q = 0;

if (d ~= 0) **Node 2**

if ( (d > 0) && (n > 0) ) **Node 3**

q = 0;

r = n;

t = d;

while (r >= t) **Node 4**

t = t \* 2; **Node 5**

end

while (t ~= d) **Node 6**

q = q \* 2;

t = t / 2;

if (t <= r) **Node 7**

r = r - t; **Node 8**

q = q + 1;

end **Node 9**

end

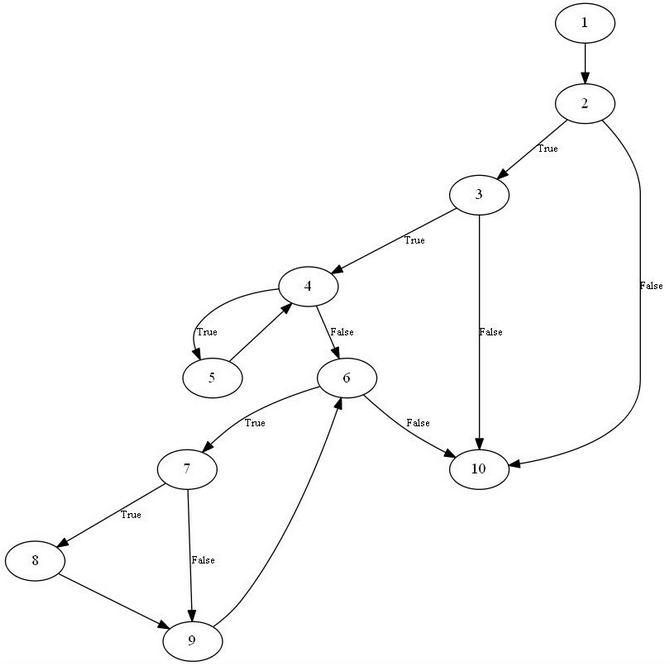
end

end

end **Node 10**

|  |  |  |
| --- | --- | --- |
| 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 [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 |

|  |  |  |
| --- | --- | --- |
| 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,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;  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 '(F) ' ];  traversedPath = [traversedPath '9 ' ];  end  end  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '10 ' ];  end |



Nodes(N) = 10  
Edges(E) = 14  
V(G) = E - N + 2  
         = 6

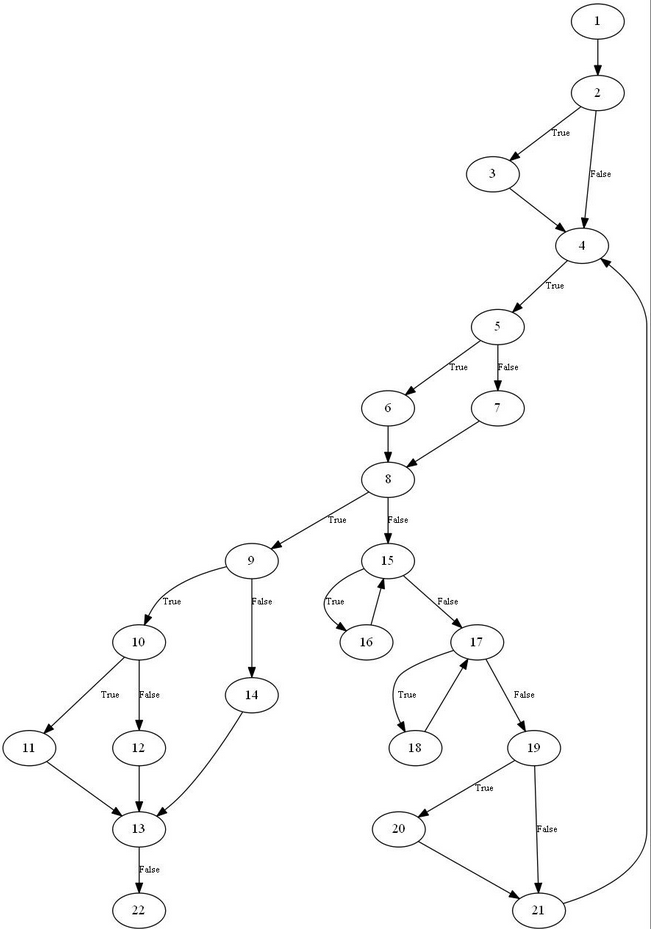
### Path

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

Lampiran 9 fB2002

|  |  |  |
| --- | --- | --- |
| 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  38  39  40  41  42  43  44  45  46  47 | **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 14**  **Node 13**  **Node 15**  **Node 16**  **Node 17**  **Node 18**  **Node 19**  **Node 20**  **Node 21**  **Node 22** | function a = findBueno2002(numbersIn)  f = numbersIn(1); % key or index  a = numbersIn(2:end); % an array of integers to be re-arranged  % n = length(numbers);  b = 0;  m = 1;  ns = length(a);  % Probe added on 02.09.2010  if f > ns  f = mod(ns,f);  end  i = 1;  while ((m < ns) || b)  if (~b)  i = m;  j = ns;  else  b = 0;  end  if (i > j)  if (f > j)  if (i > f)  m = ns;  else  m = i;  end  else  ns = j;  end  else  while (a(i) < a(f))  i = i + 1  end  while (a(f) < a(j))  j = j - 1 ;  end  if (i <= j)  w = a(i);  a(i) = a(j);  a(j) = w;  i = i + 1;  j = j - 1;  end  b = 1;  end  end  end |

|  |  |  |
| --- | --- | --- |
| 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 14**  **Node 13**  **Node 15**  **Node 16**  **Node 17**  **Node 18**  **Node 19**  **Node 20**  **Node 21**  **Node 22** | function [traversedPath,a] = findBueno2002(numbersIn)  traversedPath = [];  traversedPath = [traversedPath '1 ' ];  f = numbersIn(1); % key or index  a = numbersIn(2:end); % an array of integers to be re-arranged  % n = length(numbers);  b = 0;  m = 1;  ns = length(a);  % Probe added on 02.09.2010  % instrument Branch # 1  traversedPath = [traversedPath '2 ' ];  if f > ns  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '3 ' ];  f = mod(ns,f);  end  i = 1;  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 2  traversedPath = [traversedPath '4 ' ];  while ((m < ns) || b)  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 3  traversedPath = [traversedPath '5 ' ];  if (~b)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '6 ' ];  i = m;  j = ns;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '7 ' ];  b = 0;  end  % instrument Branch # 4  traversedPath = [traversedPath '8 ' ];  if (i > j)  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 5  traversedPath = [traversedPath '9 ' ];  if (f > j)  traversedPath = [traversedPath '(T) ' ];  % instrument Branch # 6  traversedPath = [traversedPath '10 ' ];  if (i > f)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '11 ' ];  m = ns;  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '12 ' ];  m = i;  end  else  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '14 ' ];  ns = j;  % instrument Branch # 7  traversedPath = [traversedPath '13 ' ];  end  else  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 8  traversedPath = [traversedPath '15 ' ];  while (a(i) < a(f))  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '16 ' ];  i = i + 1  end  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 9  traversedPath = [traversedPath '17 ' ];  while (a(f) < a(j))  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '18 ' ];  j = j - 1 ;  end  traversedPath = [traversedPath '(F) ' ];  % instrument Branch # 10  traversedPath = [traversedPath '19 ' ];  if (i <= j)  traversedPath = [traversedPath '(T) ' ];  traversedPath = [traversedPath '20 ' ];  w = a(i);  a(i) = a(j);  a(j) = w;  i = i + 1;  j = j - 1;  end  b = 1;  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '21 ' ];  end  traversedPath = [traversedPath '(F) ' ];  traversedPath = [traversedPath '22 ' ];  end |



Nodes(N) = 22  
Edges(E) = 29  
V(G) = E - N + 2  
         = 9

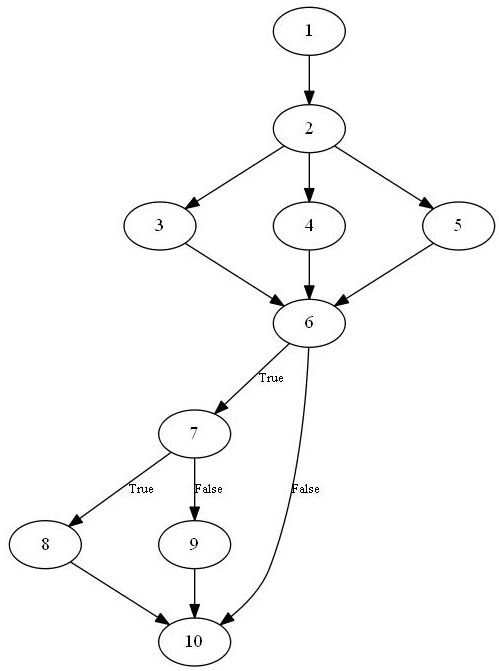
### Path

1. 1 2 (T) 3 4 (T) 5 (T) 6 8 (T) 9 (T) 10 (T) 11 13 (F) 22
2. 1 2 (T) 3 4 (T) 5 (T) 6 8 (T) 9 (T) 10 (F) 12 13 (F) 22
3. 1 2 (T) 3 4 (T) 5 (T) 6 8 (T) 9 (F) 14 13 (F) 22
4. 1 2 (T) 3 4 (T) 5 (F) 7 8 (T) 9 (T) 10 (T) 11 13 (F) 22
5. 1 2 (T) 3 4 (T) 5 (F) 7 8 (T) 9 (T) 10 (F) 12 13 (F) 22
6. 1 2 (T) 3 4 (T) 5 (F) 7 8 (T) 9 (F) 14 13 (F) 22
7. 1 2 (F) 4 (T) 5 (T) 6 8 (T) 9 (T) 10 (T) 11 13 (F) 22
8. 1 2 (F) 4 (T) 5 (T) 6 8 (T) 9 (T) 10 (F) 12 13 (F) 22
9. 1 2 (F) 4 (T) 5 (T) 6 8 (T) 9 (F) 14 13 (F) 22
10. 1 2 (F) 4 (T) 5 (F) 7 8 (T) 9 (T) 10 (T) 11 13 (F) 22
11. 1 2 (F) 4 (T) 5 (F) 7 8 (T) 9 (T) 10 (F) 12 13 (F) 22
12. 1 2 (F) 4 (T) 5 (F) 7 8 (T) 9 (F) 14 13 (F) 22

Lampiran 10 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 |

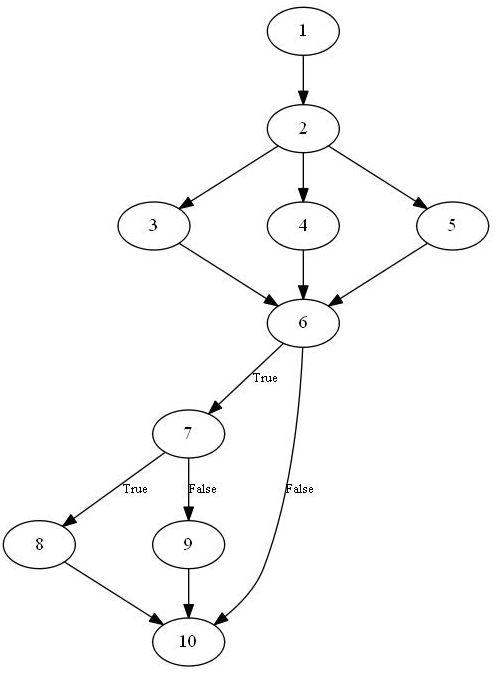
|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | **Node 1**  **Node 2**  **Node 3**  **Node 4**  **Node 5**  **Node 6**  **Node 7**  **Node 8** | function [traversedPath,branchVal] = fitnessMiniMaxi(branchNo, predicate)  traversedPath = [];  traversedPath = [traversedPath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



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

### Path

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



**RIWAYAT HIDUP**

Dalam riwayat hidup dijelaskan tempat dan tanggal kelahiran mahasiswa, putra dan putri ke berapa dari orang tua, nama kedua orang tua atau wali. Untuk skripsi, tuliskan pendidikan penulis seja\k sekolah menengah hingga terdaftar sebagai mahasiswa IPB. Kegiatan penulis di luar akademik yang menunjang pendidikan juga baik dicantumkan, terutama prestasi akademik yang pernah diraih selama masa kemahasiswaan. Uraian tentang riwayat hidup tidak lebih dari satu halaman.