PENGGUNAAN *ANT COLONY OPTIMIZATION* UNTUK PEMBANGKITAN DATA TES SECARA OTOMATIS DALAM PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh :
WIRANDA AVIV ALFIAN
NIM. 15650062

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2022

PENGGUNAAN *ANT COLONY OPTIMIZATION* UNTUK PEMBANGKITAN DATA TES SECARA OTOMATIS DALAM PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Diajukan kepada:
Universitas Islam Negeri Maulana Malik Ibrahim Malang
Untuk memenuhi Salah Satu Persyaratan dalam
Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :
WIRANDA AVIV ALFIAN
NIM. 15650062

JURUSAN TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM MALANG
2022

HALAMAN PERSETUJUAN

PENGGUNAAN ANT COLONY OPTIMIZATION UNTUK PEMBANGKITAN DATA TES SECARA OTOMATIS DALAM PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh :
WIRANDA AVIV ALFIAN
NIM. 15650062

Telah Diperiksa dan Disetujui untuk Diuji

Tanggal : 10 Juni 2022

Dosen Pembimbing I	Dosen Pembimbing II
Fatchurrohman, M.Kom	Ajib Hanani, M.T
NIP. 19700731 200501 1 002	NIDT. 19840731 20160801 1 076

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Faehrul Kurniawan ST., M.MT ., IPM
NIP. 19771020 200912 1 001

HALAMAN PENGESAHAN

PENGGUNAAN *ANT COLONY OPTIMIZATION* UNTUK PEMBANGKITAN DATA TES SECARA OTOMATIS DALAM PENGUJIAN PERANGKAT LUNAK

SKRIPSI

Oleh :
WIRANDA AVIV ALFIAN
NIM. 15650062

Telah Dipertahankan di Depan Dewan Penguji Skripsi
dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan
Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal : 16 Juni 2022

Susunan Dewan Penguji

Penguji Utama	:	Agung Teguh Wibowo Almais, M.T NIP. 19860103 20180201 1 235	( )
Ketua Penguji	:	Hani Nurhayati, M. T NIP. 19780625 200801 2 006	( )
Sekretaris Penguji	:	Fatchurrohman, M.Kom NIP. 19700731 200501 1 002	( )
Anggota Penguji	:	Ajib Hanani, M.T NIDT. 19840731 20160801 1 076	( )

Mengetahui,
Ketua Jurusan Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang

Dr. Faehrul Kurniawan ST., M.MT ., IPM
NIP. 19771020 200912 1 001

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama	: Wiranda Aviv Alfian
NIM	: 15650062
Fakultas / Jurusan	: Sains dan Teknologi / Teknik Informatika
Judul Skripsi	: Penggunaan Ant Colony Optimization untuk Pembangkitan Data Tes Secara Otomatis dalam Pengujian Perangkat Lunak

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini merupakan hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 28 Juni 2022
Yang membuat pernyataan,

Wiranda Aviv Alfian
NIM. 15650062

HALAMAN MOTTO

"Love with your heart, use your head for everything else"

HALAMAN PERSEMBAHAN

اَلْحَمْدُ للَّهِ رَبِّ الْعالَمِينَ

Puji syukur atas kehadirat Allah SWT, dengan mengucap Alhamdulillah penulis mempersembahkan sebuah karya untuk orang-orang yang sangat berarti.

Terima kasih penulis ucapkan kepada kedua Orang Tua yang selalu memberikan cinta, kasih sayang, motivasi, doa, harapan dan pengorbanan yang luar biasa untuk memberikan yang terbaik dalam segala hal di kehidupan penulis. Ibu Khoirun Nikma dan Bapak Daroini. Karena jerih payah dan keringat beliau, penulis bisa mencapai tahap ini. Tidak lupa adik dan saudara-saudara yang lainnya yang selalu memberikan motivasi serta arahan untuk menuntaskan pendidikan dan membuka tahap selanjutnya dalam hidup.

Terima kasih pula penulis ucapkan kepada Bapak Fatchurrohman selaku dosen pembimbing pertama dan Bapak Ajib Hanani selaku pembimbing kedua yang telah membimbing dalam melakukan penilitian ini dan memberikan motivasi serta dorongan hingga penelitian ini dapat terselesaikan dengan lancar.

Tidak lupa terima kasih penulis ucapkan kepada rekan-rekan seperjuangan jurusan Teknik Informatika 2015 UIN Maulana Malik Ibrahim Malang yang telah menemani, menyemangati, dan mau untuk direpoti dengan pertanyaan-pertanyaan seputar penelitian ini.

Terima kasih juga untuk orang-orang yang tidak dapat disebutkan satu per satu yang telah memberikan motivasi, arahan dan doa yang tiada henti hingga penelitian ini dapat terselesaikan.

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيْم

Alhamdulillah rabbil 'alamiin, segala puji bagi Allah SWT, yang telah memberikan karunia, rahmat dan hidayahnya. Sehingga memberikan kemudahan dalam proses penyusunan skripsi dengan judul "**Penggunaan Ant Colony Optimization untuk Pembangkitan Data Tes Secara Otomatis dalam Pengujian Perangkat Lunak**" dengan lancar dan baik. Sholawat serta salam semoga senantiasa tersampaikan kepada Nabi Muhammad SAW yang memberikan syafaat dari zaman jahiliyah menuju zaman yang penuh berkah.

Penulis menyadari banyak keterbatasan yang penulis miliki, sehingga banyak pihak yang telah memberikan bantuan baik moril maupun materil dalam proses menyelesaikan penelitian ini. Maka dari itu dengan segenap kerendahan hati penulis mengucapkan terima kasih kepada :

1. Prof. Dr. H. M. Zainuddin, MA selaku Rektor UIN Maulana Malik Ibrahim Malang.
2. Dr. Sri Harini M.Si selaku Dekan Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
3. Dr. Fachrul Kurniawan ST., M.MT ., IPM selaku Ketua Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.
4. Bapak Fatchurrohman, M.Kom selaku dosen pembimbing pertama dan Bapak Ajib Hanani, M.T selaku dosen pembimbing kedua yang senantiasa sabar dan berkenan meluangkan waktunya untuk membimbing dan memberikan arahan.
5. Seluruh dosen dan staf jurusan Teknik Informatikan Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang yang telah memberikan ilmu dan pengalaman yang bermanfaat.
6. Segenap civitas akademik Jurusan Teknik Informatika Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

7. Kedua Orang Tua dan seluruh keluarga besar yang senantiasa mendukung dan mendoakan.
8. Rekan-rekan dan sahabat seperjuangan Jurusan Teknik Informatika 2015 Fakultas Sains dan Teknologi UIN Maulana Malik Ibrahim Malang.

Penulis menyadari dalam karya ini masih banyak kekurangan. Oleh karena itu penulis selalu menerima segala kritik dan saran dari pembaca. Semoga karya ini dapat bermanfaat dan dipergunakan mestinya bagi seluruh pihak.

Malang, 27 Juni 2022

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

ABSTRAK

Alfian, Wiranda Aviv. 2022. ***Penggunaan Ant Colony Optimization untuk Pembangkitan Data Tes Secara Otomatis Dalam Pengujian Perangkat Lunak.*** Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Fatchurrohman, M.Kom (II) Ajib Hanani, M.T.

Kata Kunci : *Ant Colony Optimization*, Data Tes, Pengujian Perangkat Lunak

Pengujian perangkat lunak memainkan peran yang penting dalam menjaga kualitas perangkat lunak. Akan tetapi, pengujian memakan cukup banyak biaya, waktu, dan tenaga. Bahkan, sering kali biaya untuk pengujian perangkat lunak lebih dari 50% dari total biaya dalam pengembangan perangkat lunak. Selain itu, Menghasilkan data tes merupakan langkah terpanjang dalam proses pengujian perangkat lunak. Oleh karena itu, pembuatan data uji dan evaluasi pengujian adalah pekerjaan yang paling menuntut dan akan sangat diuntungkan dari otomatisasi. Untuk mendapatkan hasil dengan akurasi yang baik diterapkan pendekatan kecerdasan buatan. Pada penelitian ini diusulkan algoritma Ant Colony Optimization (ACO) sebagai pembangkit data uji untuk menjalankan semua cabang suatu program. *Control flow graph* dibangkitkan dari sebuah kode program untuk menggambarkan aliran kode program tersebut. Kemudian, data tes yang sesuai untuk setiap jalur percabangan dihasilkan. Dengan adanya data tes yang akurat, maka kesalahan-kesalahan pada perangkat lunak yang diuji dapat segera diketahui.

ABSTRACT

Alfian, Wiranda Aviv. 2022. ***Application of Ant Colony Optimization for Automatic Test Data Generation in Software Testing.*** Essay. Department of Informatics, Faculty of Science and Technology, Maulana Malik Ibrahim State Islamic University of Malang. Counselor: (I) Fatchurrohman, M.Kom (II) Ajib Hanani, M.T.

Keyword : *Ant Colony Optimization, Test Data, Software Testing*

Software testing plays an important role in maintaining software quality. However, testing costs a lot of money, time and effort. In fact, often the cost of software testing is more than 50% of the total cost of software development. In addition, Generating test data is the longest step in the software testing process. Therefore, test data generation and test evaluation are the most demanding jobs and will greatly benefit from automation. To get results with good accuracy, an artificial intelligence approach is applied. In this study, the Ant Colony Optimization (ACO) algorithm is proposed as a test data generator to run all branches of a program. The control flow graph is generated from a program code to describe the flow of the program code. Then, the appropriate test data for each branching path is generated. With accurate test data, errors in the software being tested can be immediately identified.

الملخص

ألفيان، ويراندا عفيف. ٢٢٠٢. **تطبيق تحسين مستعمرة النمل (Ant Colony Optimization) لتوليد بيانات الاختبار التلقائي في اختبار البرمجيات**. الجث العلمي. قسم تقنيّة المعلومات، كلية العلوم والتكنولوجيا، جامعة مولانا مالك إبراهيم الإسلامية الحككومية بملانج. المشرف : ١) فتح الرحمن الماجستير، ٢) أجب حنان المااجستير.

الكلمات الرئيسية : تحسين مستعمرة النمل ، بيانات الاختبار ، اختبار البرمجيات

يلعب اختبار البرامج دورًا مهمًا في الحفاظ على جودة البرامج. ومع ذلك ، فإن الاختبار يتطلب الكثير من الوقت والمال. في الواقع ، غالبًا ما يتطلب أكثر من ٥٠٪ من التكلفة الإجمالية لتطوير البرمجيات. يعد إنشاء بيانات الاختبار المرحلة الأكثر استهلاكا للوقت والأكثر استهلاكا للتكلفة في اختبار البرامج. وبالتالي ، فإن إنشاء بيانات الاختبار وتقييم الاختبار هما الأكثر تطلبًا وسيستفيدان بشكل كبير من الأتمتة. من أجل الحصول على نتائج بدقة جيدة ، يتم تطبيق نهج ذكاء اصطناعي. في هذه الدراسة ، تم اقتراح خوارزمية تحسين مستعمرة النمل ((Ant Colony Optimization (ACO)كمولد بيانات اختبار لتنفيذ جميع الفروع في البرنامج. يتم إنشاء الرسم البياني للتحكم في التدفق من رمز برنامج لوصف تدفق رمز البرنامج. بعد ذلك ، بيانات الاختبار المناسبة لكل مسار متفرع. باستخدام بيانات الاختبار الدقيقة ، يمكن العثور على أخطاء البرامج على الفور.

BAB I

PENDAHULUAN

Bab ini akan menjelaskan mengenai latar belakang penelitian, identifikasi masalah, tujuan penelitian, manfaat penelitian dan batasan penelitian. Latar belakang penelitian berisi penjelasan mengenai alasan peneliti mengangkat permasalaha penelitian ini. Identifikasi masalah berisi sebuah pertanyaan yang didasarkan pad latar belakang penelitian ini. Tujuan penelitian berisi tujuan dilakukannya penelitian ini dan batasan penelitian berisi batasan pada penelitian agar penelitian tidak meluas dari fokus dilakukannya penelitian ini.

1.1 Latar Belakang

Pengujian perangkat lunak memegang peranan penting dalam menjaga kualitas perangkat lunak. Kegagalan dalam melakukan pengujian perangkat lunak dapat menyebabkan produk yang dihasilkan tidak berjalan dengan baik dan dapat menimbulkan kerugian yang besar. Namun, pengujian memerlukan waktu dan biaya yang tidak sedikit. Bahkan, seringkali memerlukan lebih dari 50% biaya keseluruhan dalam pengembangan perangkat lunak (Maulana et al., 2015). Tujuan utama dari pengujian adalah untuk mendeteksi kesalahan-kesalahan perangkat lunak sehingga bisa ditemukan dan diperbaiki. Karena terbatasnya waktu dan biaya, maka tidak memungkinkan untuk melakukan pengujian secara manual dan memperbaiki kesalahannya (Srivastava & Baby, 2010). Oleh karena itu, untuk menghemat waktu dan biaya diperlukan pengujian perangkat lunak secara otomatis.

Pengujian perangkat lunak terdiri dari 3 aktivitas utama yaitu : pembangkitan data tes, pelaksanaan tes, dan evaluasi hasil tes. Dari ketiga

aktivitas tersebut, pembangkitan data tes dan mengevaluasi pengujian adalah yang paling banyak memakan waktu dan biaya (Li & Lam, 2007). Sehingga, pembangkitan data tes dan evaluasi pengujian adalah yang paling memerlukan dan akan mendapat banyak keuntungan dari otomatisasi. Pengujian perangkat lunak secara otomatis diharapkan dapat meningkatkan efisiensi proses pengujian.

Dalam kaitannya dengan Al-Qur'an, Allah SWT juga menjelaskan dalam firman-nya :

الَّذِي خَلَقَ الْمَوْتَ وَالْحَيَاةَ لِيَبْلُوَكُمْ أَيُّكُمْ أَحْسَنُ عَمَلًا ۚ وَهُوَ الْعَزِيزُ الْغَفُورُ

Artinya : *"Yang menjadikan mati dan hidup, supaya Dia menguji kamu, siapa di antara kamu yang lebih baik amalnya. Dan Dia Maha Perkasa lagi Maha Pengampun". (QS Al-Mulk 67:02)*

Tafsir Ibnu Katsir menjelaskan tentang *QS Al-Mulk* pada potongan ayat. (الَّذِي خَلَقَ الْمَوْتَ وَالْحَيَاة) "*Yang menjadikan mati dan hidup*" Ayat ini dijadikan oleh orang-orang yang berpendapat bahwa kematian adalah sesuatu yang wujud karena ia diciptikan (makhluk). Sedangkan makna ayat itu sendiri bahwa Allah telah mengadakan makhluk ini dari ketiadaan untuk menguji mereka, yakni untuk menguji siapakah diantara mereka yang paling baik amalnya. Sebagaimana yang difirmankan Allah Ta'ala,

كَيْفَ تَكْفُرُونَ بِاللَّهِ وَكُنْتُمْ أَمْوَاتًا فَأَحْيَاكُمْ ۖ

Artinya:

"*Mengapa kamu kafir kepada Allah, padahal kamu tadinya mati, lalu Allah meghidupkan kamu,*" (*QS. Al-Baqarah 2:28*).

Dengan demikian, keadaan pertama, yaitu ketiadaan sebagai maut (kematian). Sedangkan penciptaan disebut hayat (kehidupan). Oleh karena itu, Allah Ta'ala (لِيَبْلُوَكُمْ أَيُّكُمْ أَحْسَنُ عَمَلًا) "*Supaya Dia mengujimu, supaya diantara kamu yang lebik baik amalnya.*" Yakni, yang paling baik amalnya, sebagaimana yang dikatakan oleh Muhammad bin Ajlan. Dan Allah tidak Mengatakan "*Yang paling banyak amalnya.*"

Allah menguji hamba-Nya untuk mengetahui yang lebih baik amal diantara hamba-hamba-Nya. Sedangkan dalam penelitian ini, pengujian dimaksudkan sebagai sarana untuk mendapatkan *software* dengan kualitas yang baik.

Saat ini, teknik kecerdasan buatan banyak digunakan untuk otomatisasi pengujian. Penggunaan pendekatan AI di bidang pengujian perangkat lunak bertujuan untuk akurasi yang lebih baik. Pendekatan yang banyak digunakan diantaranya adalah *Ant Colony Optimization* (ACO), *Genetic Algorithm* (GA), *Tabu Search* (TB), dan lain-lain. Pendekatan ini dikenal sebagai pendekatan metaheuristik (Srivastava & Baby, 2010).

Dalam penelitian ini, program yang diuji adalah program benchmark publik yang juga digunakan oleh peneliti lain seperti oleh Pachauri & Srivastava(2013) dan Ferrer et al(2012). Program benchmark tersebut adalah Myers *triangle*, Michael *triangle*, dan Sthamer *triangle*. Penggunaan program benchmark publik bertujuan agar hasil penelitian lebih mudah divalidasi. Peneliti mengambil judul Penggunaan Ant Colony Optimization untuk Pembangkitan Data Tes dalam Pengujian Perangkat Lunak Otomatis sebagai penelitian yang akan dilakukan dalam skripsi ini.

1.2 Identifikasi Masalah

Berdasarkan uraian latar belakang penelitian, maka masalah yang akan diangkat pada penelitian ini adalah :

1. Bagaimana cara membangkitkan data tes secara otomatis dengan menggunakan algoritma *ant colony*.
2. Berapa lama waktu yang diperlukan algoritma ant colony untuk membangkitkan data tes dari setiap program *benchmark.*

1.3 Tujuan

Berdasarkan identifikasi masalah, maka tujuan dari penelitian ini adalah :

1. Membangkitkan data tes secara otomatis dengan menggunakan algoritma *ant colony*.
2. Mengukur waktu yang dibutuhkan untuk membangkitkan data tes pada setiap program *benchmark.*

1.4 Batasan Masalah

Batasan penelitian ini adalah program yang diuji adalah program *benchmark* publik yang banyak digunakan oleh peneliti lain, yaitu Myers *triangle*, Michael *triangle*, dan Sthamer *triangle*.

1.5 Manfaat

Hasil penelitian ini dapat digunakan *software tester* untuk menghemat waktu dan tenaga yang dibutuhkan untuk melakukan pengujian perangkat lunak.

1.6 Sistematika Penulisan

Laporan penelitian ini terdiri dari lima bab, dimana isi dari setiap bab terdiri dari :

BAB I : PENDAHULUAN

Berisi tentang latar belakang dari masalah yang akan diteliti, tujuan dan manfaat penelitian dari penelitian, batasan masalah pada penelitian, metodologi penelitian, serta sistematika penulisan laporan penelitian.

BAB II: TINJAUAN PUSTAKA

Bab ini berisi penjelasan mengenai penelitian yang telah dilakukan ataupun teori dasar dan data-data yang terkait dengan pembangkitan data tes dalam pengujian perangkat lunak.

BAB III : METODE PENELITIAN

Bab ini berisi metode penelitian yang menjelaskan bagaimana penelitian ini dijalankan. Meliputi hasil analisa dan rincian langkah yang digunakan dalam pembangkitan data tes dalam pengujian perangkat lunak dengan menerapkan metode *Ant Colony Optimization.*

BAB IV : HASIL DAN PEMBAHASAN

Bab ini berisi uji coba dari aplikasi yang telah dibuat dan dilakukan pembahasan secara terperinci terhadap proses pembangkit data tes. Untuk selanjutnya membahas tentang hasil yang di dapat dari mengimplementasikan *Ant Colony Optimization.*

BAB V : KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan dari penelitian yang telah dilakukan, saran dan kritik dari penelitian agar dapat dikembangkan pada penelitian selanjutnya.

BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terkait

Hasil penelitian yang telah dilakukan dan berkaitan dengan penelitian ini adalah sebagai berikut :

1. Anand et al. (2013) melakukan survey terhadap beberapa teknik yang sering digunakan untuk membangkitkan data tes secara otomatis, diantaranya adalah *symbolic execution, model-based*, kombinatorial, *adaptive random, search-based*. *Search Based Software Testing* (SBST) adalah cabang dari *Search Based Software Engineering* (SBSE) yang merumuskan kembali tujuan pengujian sebagai fungsi *fitness* untuk pencarian otomatis. Pendekatan ini sangat umum dan sangat dapat diterapkan secara luas karena setiap tujuan tes yang dapat diukur adalah kandidat untuk transformasi ini menjadi fungsi *fitness*.
2. Srivastava & Baby (2010) dalam penelitiannya menjelaskan tentang metode standar pengujian berbasis transisi dan tingkat cakupannya. Karena metode yang ada menggunakan GA tidak menyediakan cakupan penuh untuk perangkat lunak apa pun, maka diusulkan model untuk cakupan berbasis transisi dengan menggunakan *ant colony optimization*. Hasil yang didapatkan dengan menerapkan metode ini sangat menggembirakan. Dengan menggunakan kekuatan pendekatan ACO, didapatkan generasi test sequence yang optimal untuk pengujian perangkat lunak berbasis-transisi.
3. Mao et al. (2014) dalam penelitiannya tentang penerapan *ant colony optimization* dalam pembangkitan data tes untuk pengujian struktural,

menjelaskan bahwa pengujian struktural banyak dipakai untuk mendeteksi potensi kesalahan dalam kode program. Namun, bagaimana menghasilkan data uji dengan kemampuan yang lebih kuat untuk mencakup elemen-elemen program masihmenjadi masalah. Dalam penelitian ini, metode *ant colony optimization* diadaptasi untuk menghasilkan data uji untuk pengujian cakupan cabang. Setelah dianalisa, ACO lebih baik daripada SA dan GA dalam kemampuan cakupan, kecepatan konvergensi dan stabilitas, dan dapat dibandingkan dengan metode berbasis PSO. Selain itu, analisis sensitivitas parameter juga diselidiki untuk menemukan pengaturan yang wajar untuk metode pembuatan data tes berbasis ACO.

4. Li & Lam (2007) melakukan penelitian tentang pembangkitan data tes menggunakan *ant colony optimization*. Penelitiannya menyajikan pendekatan ACO untuk menguji pembuatan urutan untuk pengujian perangkat lunak berbasis state. Grafik dinamis terarah dibuat untuk mewakili struktur model *statechart* dari sistem perangkat lunak yang sedang diuji. Dengan menggunakan algoritma ACO yang dikembangkan, sekelompok semut dapat secara efektif menjelajahi *graph* dan menghasilkan data uji yang optimal untuk mencapai persyaratan cakupan tes.

2.2 Data

Irwansyah dan Moniaga (2014) menyebutkan bahwa data komputer adalah informasi yang diproses atau disimpan oleh komputer. Informasi ini dapat dalam bentuk dokumen teks, gambar, klip audio, program perangkat lunak, atau jenis data lainnya. Data komputer dapat diproses oleh CPU komputer dan disimpan dalam *file* dan *folder* di *harddisk* komputer. Pada tingkat paling dasar, data

komputer adalah sekelompok satu dan nol, yang dikenal sebagai data biner. Karena semua data komputer dalam format biner, itu dapat dibuat, diproses, disimpan, dan disimpan secara digital. Format ini memungkinkan data untuk ditransfer dari satu komputer ke komputer lain menggunakan koneksi jaringan atau berbagai perangkat media tanpa kehilangan kualitasnya.

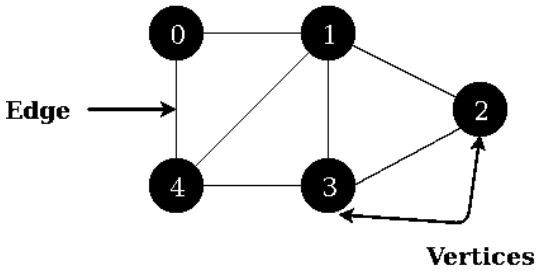
2.3 Data Tes

Data tes adalah semua data yang digunakan dalam pengujian sistem perangkat lunak. Untuk menguji aplikasi perangkat lunak, perlu dimasukkan beberapa data untuk menguji sebagian besar fitur. Data tes digunakan untuk melakukan konfirmasi terhadap hasil yang diharapkan. Misalnya ketika sejumlah data tes dimasukkan, apakah hasil yang diharapkan sesuai dengan yang ditampilkan dan apa yang terjadi jika program mendapat input yang salah. Data tes dapat dihasilkan secara manual oleh penguji atau menggunakan *tool* otomatis yang mendukung pengujian (Offut et al., 1999).

Data tes biasanya berupa input dari suatu program. Data tes dapat berupa lembar *excel* yang dapat dimasukkan secara manual saat menjalankan uji kasus atau dapat dibaca secara otomatis dari file (XML, *flat file*, basis data, dan lain-lain) menggunakan *tool*.

2.4 Graph

Graph merupakan struktur data non-linear yang terdiri kumpulan dari *node* dan *edge* (Rerung, 2020). *Node* juga disebut sebagai simpul dan *edge* adalah garis atau busur yang menghubungkan dua *node* dalam *graph*. Jadi, *Graph* terdiri dari sekumpulan simpul/*node* dan beberapa *edge* yang menghubungkan sepasang *node*.



Gambar 2. 1 Contoh *graph*

Sebagai contoh, pada gambar 2.1 dapat dilihat bahwa himpunan *node* V = {0,1,2,3,4} dan himpunan *edge* E = {01, 12, 23, 34, 04, 14, 13}.

Graph dapat digunakan untuk memecahkan banyak masalah kehidupan nyata. *Graph* biasa digunakan untuk mewakili jaringan, seperti jalur di kota, jaringan telepon atau jaringan sirkuit. *Graph* juga digunakan di jejaring sosial seperti linkedIn, Twitter, dan Facebook. Misalnya, pada Facebook, setiap orang diwakili dengan *node*. Setiap node adalah struktur dan berisi informasi seperti id orang, nama, jenis kelamin, lokal dan lain-lain.

2.5 Software Testing

Software testing merupakan aktivitas yang dilakukan untuk mengevaluasi dan meningkatkan kualitas suatu produk perangkat lunak, dengan cara mengidentifikasi bug serta permasalahan yang terdapat pada produk tersebut (Utting, 2007). Software testing berperan dalam menentukan ukuruan suatu project. Ukuran kualitas umumnya dibatasi oleh *correctness* (kebenaran), *completeness* (kesempurnaan), dan *security* (keamanan). Namun ukuran kualitas menurut ISO adalah *reliability* (keandalan), *efficiency* (efisiensi), *portability* (portabilitas), *maintainability* (pemeliharaan), *compatibility* (kesesuaian), dan *usability* (kegunaan).

2.6 Path Testing

Pengujian *path* adalah metode pengujian struktural yang melibatkan penggunaan *source code* suatu program untuk menemukan setiap *path* yang dapat dieksekusi untuk membantu untuk menemukan semua kesalahan terletak di dalam kode (Latiu, et al., 2007). Metode ini dirancang untuk mengeksekusi semua atau jalur yang dipilih melalui program komputer. Setiap program perangkat lunak terdiri dari beberapa titik masuk dan keluar. Menguji setiap poin ini cukup berat dan menyita waktu. Untuk itu, digunakan basis path testing untuk mengurangi tes yang berlebihan dan untuk mencapai cakupan tes maksimum.

Basis path testing mendefinisikan test case berdasarkan *flow* atau *logical path* yang dapat diambil melalui program. Dalam rekayasa perangkat lunak, *basis path testing* dilakukan dengan menjalankan semua blok dalam suatu program untuk mencapai cakupan jalur maksimum dengan jumlah *test case* yang paling sedikit. Tujuan dari *basis path testing* dalam pengujian perangkat lunak adalah menentukan jumlah jalur independen, sehingga jumlah *test case* yang diperlukan dapat didefinisikan secara eksplisit (memaksimalkan cakupan setiap *test case*).

2.7 Automated Testing

Automated testing merupakan proses pengujian yang digunakan untuk mempermudah proses dan dokumentasi pengujian, serta mengefektifkan proses pengeksekusian dan pengukuranjpada pengujian. *Automated testing* akan sangat terasa manfaatnya (peningkatan efisiensi biaya dan efektifitas sumber daya) dalam *regression testing*. Terbatasnya waktu merupakan hambatan terbesar dalam melakukan regression testing, sehingga pada testing secara manual jumlah *test case* untuk *regression testing* dibatasi hanya 10% dari jumlah tes case yang

dilakukan pada awal testing. Berdasarkan pada studi yang dilakukan oleh Software Engineering Institute – Bellcore Study, terdapat kecenderungan terjadinya defect/bug baru setelah dilakukan perubahan atau perbaikan pada sistem (lebih dari 60%) atau error baru akan muncul setiap 6 baris kode dirubah.

Faktor-faktor yang mendukung berkembangnya pengujian perangkat lunak diantaranya adalah penghematan biaya pengembangan, durasi pengujian yang dipersingkat, peningkatan kecermatan dalam pelaksanaan pengujian, dan peningkatan hasil pengujian (Rina, 2009).

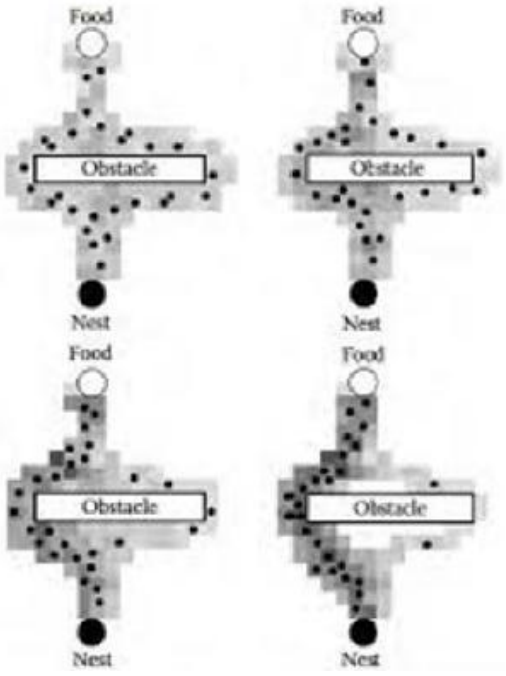
Keuntungan-keuntungan dari pengujian otomatis antara lain:

a. Meningkatkan produktivitas
b. Menghemat uang
c. Meningkatkan kualitas perangkat lunak
d. Mengurangi waktu pengujian
e. Mendukung berbagai aplikasi
f. Meningkatkan cakupan pengujian
g. Pengurangan pekerjaan yang berulang-ulang

2.8 Ant Colony Optimization

Ant Colony Optimization termasuk dalam kelompok *Swarm Intelligence*, yang merupakan salah satu jenis pengembangan paradigma yang digunakan untuk menyelesaikan masalah optimasi dimana inspirasi yang digunakan untuk memecahkan masalah tersebut berasal dari perilaku kumpulan atau kawanan (*swarm*) serangga. ACO biasanya digunakan untuk menyelesaikan *discrete optimization problems* dan persoalan yang kompleks dimana terdapat banyak variabel (Karjono et al., 2016).

Sesuai dengan namanya *ant colony optimization* terinspirasi oleh perilaku koloni semut. Algoritma ini meniru bagaimana semut berinteraksi satu dengan lainnya agar dapat menemukan sumber makanan dan membawanya ke koloni mereka dengan efisien. Selama berjalan tiap semut mengeluarkan feromon, dimana semut lainnya sensitif dengan feromon tersebut sehingga semut lain dapat untuk mengikuti jejaknya dan tercipta jalur feromon. Dalam memilih jalur, semut cenderung memilih jalur dengan konsentrasi feromon yang lebih tinggi. Jalur yang lebih pendek akan lebih sering dilewati, sehingga konsentrasi feromonnya menjadi semakin tinggi. Seiring berjalannya waktu, jalur yang lain akan memudar atau menguap. Jalur-jalur pendek akan mempunyai ketebalan feromon dengan probabistik tinggi dan membuat jalur tersebut akan dipilih dan jalur yang panjang ditinggalkan.



Gambar 2. 2Ilustrasi perubahan konsentrasi feromon

Cara kerja algoritma ant colony adalah sebagai berikut :

1. Pada awalnya, semut-semut berjalan secara acak untuk mencari makanan.

2. Ketika semut-semut menemukan persimpangan atau ketika terdapat halangan di jalur menuju makanan, mereka akan memilih jalur yang berbeda secara acak.
3. Sebagian semut akan memilih jalur kanan dan sebagian lainnya memilih jalur kiri.
4. Ketika menemukan makanan, mereka akan kembali ke sarangnya sambil meninggalkan jejak feromon.
5. Karena jalur kiri lebih pendek, semut-semut yang melewati jalur ini melakukan *rouund-trip* lebih sering. Sehingga, konsentrasi feromon pada jalur ini lebih tinggi daripada konsentrasi feromon di jalur yang panjang.
6. Feromon yang berkonsentrasi tinggi pada akhirnya akan menarik semut-semut lain untuk berpindah jalur, menuju jalur paling optimal, sedangkan jalur lainnya akan ditinggalkan.
7. Pada akhirnya, semua semut yang tadinya menempuh jalur yang berbeda-beda akan beralih ke sebuah jalur tunggal yang merupakan jalur terpendek dari sarang menuju ke tempat makanan.

Keuntungan dari optimasi *ant colony* adalah :

- Dapat mencari di antara populasi secara paralel
- Dapat menemukan solusi terbaik dengan sangat cepat
- Fleksibel, dapat beradaptasi dengan perubahan-perubahan seperti ketika jarak antar node mengalami perubahan
- Memiliki jaminan terhadap konvergensi

Kekurangan dari optimasi *ant colony* adalah :

- Distribusi probabilitas dapat berubah-ubah untuk setiap iterasi

▪ Memiliki analisis teoritis yang cenderung sulit

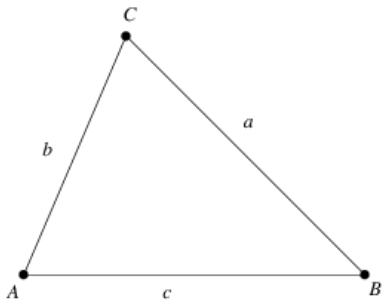
▪ Memiliki *sequence* yang bergantung pada keputusan tertentu

▪ Lebih banyak memerlukan penelitian secara eksperimental daripada penelitian secara teoritis

▪ Waktu yang dibutuhkan untuk konvergensi tidak pasti

2.9 Segitiga

Segitiga adalah poligon yang memiliki 3 buah sisi atau biasanya disebut sebagai trigon (Weisstein, 2021). Setiap segitiga memiliki tiga sisi dan tiga sudut. Umumnya, sudut-sudut pada segitiga diberi label dengan huruf besar, misalnya A,B,C. Sedangkan, sisi-sisinya diberi label dengan huruf kecil sesuai dengan sudut yang menghadap ke sisi tersebut, misalnya a,b,c. Untuk menamai suatu segitiga, disebutkan label sisinya dengan urutan berlawanan arah jarum jam. Sebagai contoh, pada gambar 2.3 dapat disebut sebagai segitiga ABC.



Gambar 2. 3Segitiga ABC

Berdasarkan besar sudut-sudutnya, segitiga dapat dikelompokkan menjadi segitiga lancip, segitiga siku-siku, dan segitiga tumpul. Segitiga lancip adalah segitiga yang semua sudutnya lancip. Segitiga siku-siku adalah segitiga yang memiliki 1 sudut siku-siku. Segitiga tumpul adalah segitiga yang memiliki 1 sudut tumpul.

Berdasarkan panjang sisi-sisinya, segitiga dapat dikelompokkan menjadi segitiga sama sisi, segitiga sama kaki, dan segitiga sembarang. Segitiga sama sisi adalah segitiga yang ketiga sisinya sama panjang. Segitiga sama kaki adalah segitiga yang memiliki 2 sisi sama panjang. Segitiga sembarang adalah segitiga yang ketiga sisinya tak sama panjang.

Untuk mengetahui apakah 3 bilangan tertentu dapat membentuk sebuah segitiga, digunakan teori ketidaksamaan segitiga. Teori ini meyatakan bahwa untuk sembarang segitiga, jumlah panjang 2 buah sisi harus lebih panjang dari panjang sisi lainnya. Misalnya, suatu segitiga memiliki sisi x,y, dan z. Jika x panjangnya 3 cm dan y panjangnya 4 cm, maka untuk membentuk segitiga, sisi z tidak boleh lebih panjang dari 7 cm.

BAB III

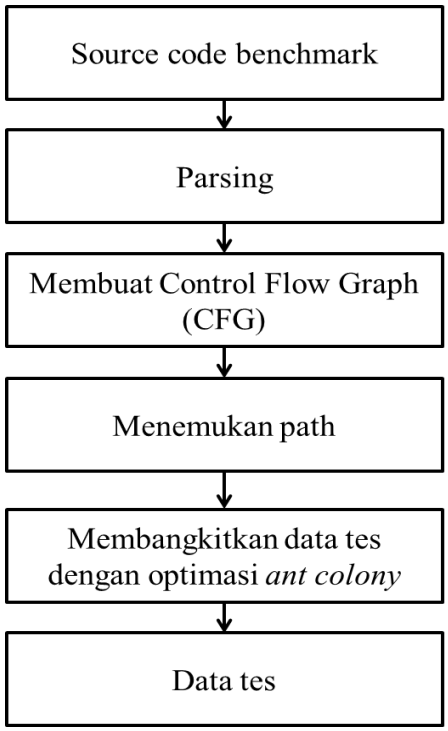
METODOLOGI PENELITIAN

3.1 Pengumpulan Data

Penelitian ini menggunakan *benchmark* data program publik yang juga digunakan oleh peneliti lain. Penggunaan *program benchmark* publik dimaksudkan agar hasil dari penelitian ini dapat divalidasi dengan hasil penelitian yang sejenis. Terdapat dua program benchmark yang digunakan dalam penelitian ini yaitu *Myers triangle* (Myers et al., 2011), *Michael triangle* (Michael et al., 2001), dan *Sthamer triangle* (Sthamer, 1995).

3.2 Perancangan Sistem

Bagian ini akan menjelaskan tentang langkah kerja dari sistem yang akan dibangun. Langkah-langkah tersebut dapat dilihat pada gambar 3.1.



Gambar 3. 1 Desain Sistem

Pada awalnya, source code diuraikan (*parsing*) untuk mengetahui arus eksekusi data. Setelah melakukan *parsing*, percabangan dianggap *node* dan arus eksekusi program dijadikan sebagai *path* untuk membentuk CFG (Manikumar et al., 2016). Dari CFG tersebut, kemudian dicari setiap *path* yang dapat mewakili berbagai kondisi pada program yang diuji. Selanjutnya, dilakukan pemabngkitan data tes yang sesuai untuk masing-masing *path*. Data tes yang dihasilkan berupa bilangan-bilangan yang akan dijadikan input program.

3.2.1 Source code benchmark

Terdapat tiga benchmark program yang digunakan dalam penelitian ini yaitu *Myers Triangle*, *Michael Triangle* dan *Sthamer Triangle*.

1. *Myers Triangle*

   Program ini menggolongkan segitiga atas dasar 3 sisi input sebagai bukan segitiga, segitiga sama kaki, segitiga sama sisi atau segitiga sembarang. *Control flow graph* yang dihasilkan memiliki 14 *node*.

2. *Michael Triangle*

   Program ini juga menggolongkan segitiga atas dasar 3 sisi input sebagai non-segitiga atau segitiga, yaitu, sama kaki, sama sisi atau sisi tak sama panjang. *Control flow graph* yang dihasilkan memiliki 26 *node*.

3. *Sthamer Triangle*

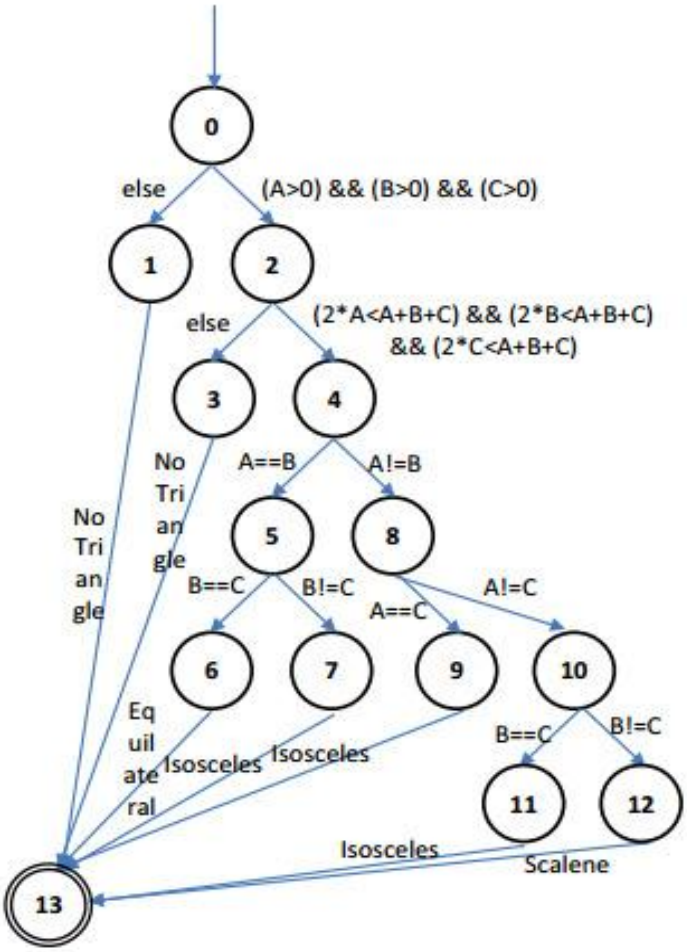
   Program ini juga menggolongkan segitiga atas dasar 3 sisi input sebagai non-segitiga atau segitiga, yaitu, sama kaki, sama sisi, segitiga siku-siku atau sisi tak sama panjang. *Control flow graph* yang dihasilkan memiliki 29 *node*.

3.2.2 Parsing program

Parsing adalah proses menganalisis serangkaian simbol dalam suatu bahasa, baik bahasa alami, bahasa komputer atau struktur data. Dalam ilmu komputer, penguraian atau parsing adalah suatu cara memecah-mecah suatu rangkaian input yang akan menghasilkan suatu pohon uraian (*parse tree*). Dalam penelitian ini, parsing digunakan untuk menemukan percabangan-percabangan yang terdapat di *program benchmark*. Pola dari percabangan tersebut nantinya akan digunakan dalam menyusun CFG.

3.2.3 Control Flow Graph

Control Flow Graph (CFG) adalah representasi grafis dari aliran kontrol atau perhitungan selama eksekusi suatu program atau aplikasi. *Control Flow Graph* terdiri dari blok-blok yang berisi *node* mulai, *node* akhir, node proses, dan *flow/arc* diantara node-node tersebut. *Control Flow Graph* dapat menggambarkan bagaimana unit program atau aplikasi dalam memproses informasi berbeda dalam sebuah sistem. Gambar 3.5 adalah contoh CFG untuk program *Myers Triangle Problem.*



Gambar 3. 2 *Control flow graph* program *Myers triangle*

3.2.4 Menentukan *path*

Path merupakan alur jalannya blok proses pada suatu perangkat lunak. Tujuan dari pengujian path adalah untuk mengeksekusi semua jalur dan menemukan cacat dalam program. Sebuah program terkadang memiliki *loop* tak terbatas yang menghasilkan jumlah *path* tak terbatas. Kompleksitas siklomatik digunakan untuk menemukan *path* independen secara linear untuk modul program (Biswas, et al., 2015). *Path* independen linear adalah *path* yang setidaknya memiliki satu *node* yang berbeda dari path-path yang sudah ditemukan. Berikut ini adaalah formula untuk menghitung kompleksitas siklomatik :

$$CC = E - N + 2P \qquad [III.1]$$

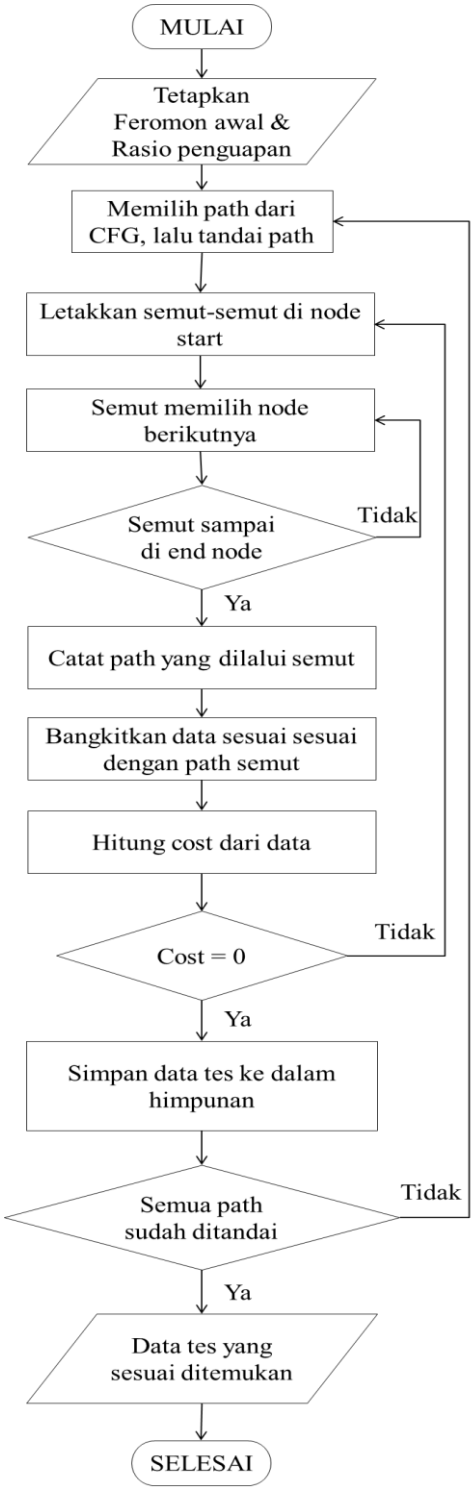
Dimana E menunjukkan jumlah *edge*, N adalah jumlah *node,* dan P adalah jumlah *exit node* pada program.

Dari CFG program Myers *triangle* pada gambar 3.2, dapat dilihat bahwa CFG tersebut memiliki 19 *edge*, 14 *node*, dan 1 *exit node*. Jadi, CC = 19-14+(2*1) = 7. Artinya, dari CFG tersebut terdapat 7 *path* yang independen secara linear. *Path-path* independen pada CFG program Myers *triangle* diantaranya adalah :

- *Path* 1 = 0 – 1 – 13
- *Path* 2 = 0 – 2 – 3 – 13
- *Path* 3 = 0 – 2 – 4 – 5 – 6 – 13
- *Path* 4 = 0 – 2 – 4 – 5 – 7 – 13
- *Path* 5 = 0 – 2 – 4 – 8 – 9 – 13
- *Path* 6 = 0 – 2 – 4 – 8 – 10 – 11 – 13
- *Path* 7 = 0 – 2 – 4 – 8 – 10 – 12 – 13

3.2.5 Pembangkitan data tes dengan optimisasi ant colony

Setiap *path* yang sudah dihasilkan harus diuji dengan menggunakan data tes yang sesuai. Untuk itu, pada penelitian ini pendekatan berbasis ACO diusulkan untuk melakukan pembangkitan data tes. Berikut ini adalah *flowchart* untuk menggambarkan langkah-langkah dalam membangkitkan data tes dengan algoritma *ant colony.*

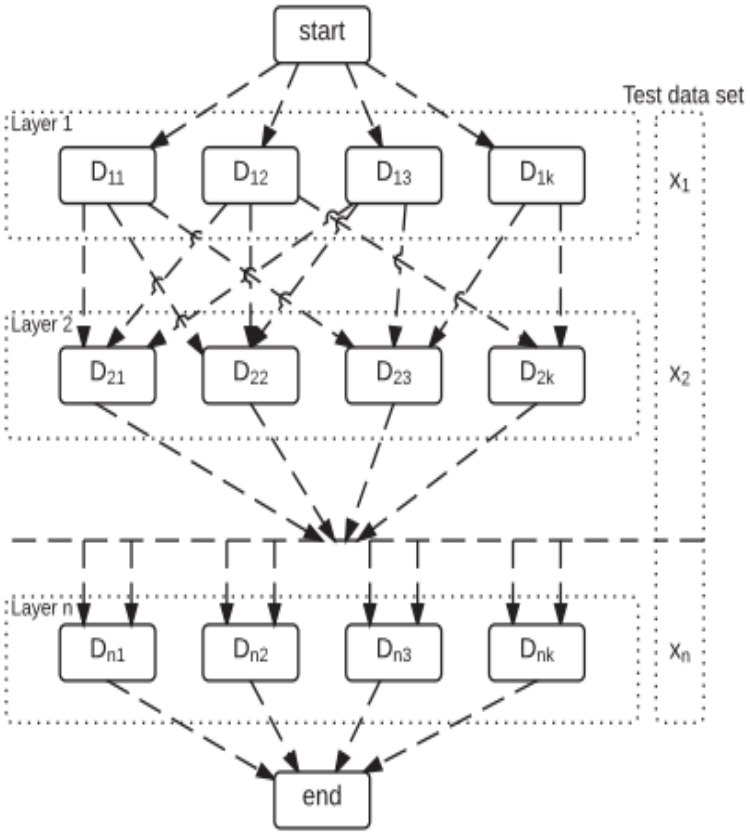


Gambar 3. 3 *Flowchart* pembangkitan data tes dengan algoritma *ant colony*

Algoritma *ant colony* terinspirasi dari cara semut mencari rute terbaik dari sarang menuju sumber makanan. Oleh karena itu, hal pertama yang dilakukan untuk memecahkan masalah dengan algoritma ant colony adalah melakukan

representasi permasalashan. Masalah direperesentasikan ke dalam bentuk pencarian rute terbaik. Sehingga, semut-semut dapat berjalan dan mencari solusi dari masalah tersebut.

Pada penelitian ini, Set input data direpresentasikan ke dalam bentuk *graph* terarah seperti pada penelitian yang dilakukan oleh Li et al.(2009) dan Biswas et al.(2015). Jumlah input pada program digambarkan sebagai lapisan (*layer*) pada *graph*. Jika suatu program memiliki jumlah input sebanyak n, maka model pencariaanya terdiri dari n *layer*. Pada setiap *layer*, terdiri dari beberapa domain yang akan mewakili bilangan-bilangan input. Setiap domain memiliki sub-domain yang memiliki aturan tertentu sehingga suatu urutan bilangan akan berada dalam satu domain.



Gambar 3. 4 *Graph* pembangkitan data tes (Biswas, et al., 2015)

Setelah membuat model pencarian, selanjutnya proses pencarian data tes akan dimulai. Pertama, beberapa semut buatan diletakkan di *start node.* Kemudian, masing-masing semut memilih node selanjutnya untuk berjalan hingga sampai pada *end node*. Ketika memilih node selanjutnya, semut-semut akan cenderung memlih node dengan jumlah feromon terbesar.

Karena sedikitnya informasi feromon di awal pencarian, algoritma *ant colony* akan mudah jatuh pada nilai optimum lokal. Untuk mengatasi masalah ini, pada tahap awal pencarian, semut-semut harus memilih jalur dengan feromon terkecil dan mengabaikan pengaruh feromon. Li et al.(2009) menyebut aturan ini sebagai strategi "memilih yang terburuk". Aturan ini hanya digunakan pada beberapa iterasi pertama saja, selanjutnya aturan akan ini diabaikan dan semut-semut kembali memilih jalur dengan jumlah feromon tertinggi. Tujuan dari strategi ini adalah agar semut-semut bisa menjelajahi jalur sebanyak mungkin dan meningkatkan cakupan pencarian. Sehingga, dapat terhindar dari kemungkinan terjebak di optimisasi lokal. Dengan demikian, aturan untuk memilih jalur adalah sebagai berikut.

```
while (m < maxnum)
      if(m <= tempnum)
            next_node(i) = min(τij);
            /* next_node(i)  = node selanjutnya yang terhubung dengan i */
            /* min(τij) = path ij dengan feromon paling sedikit */
      if(tempnum < m < maxnum)
            next_node(i) = max(τij);
            /* max(τij) = path ij dengan feromon paling banyak */
      m++;
end
/* tempnum = iterasi yang  menggunakan strategi memilih yang terburuk*/
/* maxnum = iterasi maksimal*/
/* m = loop counter */
```

Gambar 3. 5 *Pseudocode* aturan untuk memilih node selanjutnya

Dengan menggunakan aturan pada gambar 3.5, semut-semut akan memilih jalur masing-masing untuk menuju *end node*. *Node* yang dilalui setiap semut pada setiap *layer* akan dicatat, sehingga akan terbentuk sebuah *path*. Dari *path* ini, dibangkitkan bilangan secara acak berdasarkan nilai dari domain yang dilewati semut di setiap layer. Bilangan-bilangan yang dihasilkan digunakan sebagai data tes sementara.

Untuk mendeskripsikan apakah data tes yang dihasilkan sudah sesuai atau tidak, digunakan nilai objektif. Setiap data tes yang dihasilkan dihitung nilai objektifnya. Nilai objektif merupakan keluaran dari fungsi objektif. Pada kasus pembangkitan data tes, Mandyartha(2017) dalam penelitiannya menggunakan fungsi *cost*. Fungsi *cost* merupakan fungsi penalti yang menggambarkan jarak cabang (*branch distance*) yang akan dihitung bila predikat cabang tidak terpenuhi. Jika solusi memenuhi predikat cabang, maka *cost* bernilai 0. Berikut ini adalah tabel fungsi *cost* ekspresi predikat cabang.

Tabel 3. 1 Fungsi cost ekspresi predikat cabang

Ekspresi predikat	Cost bila tidak memenuhi ekspresi predikat
$a \leq b$	a – b
$a < b$	a – b + k
$a = b$	abs (a-b)
$a \neq b$	k – abs (a-b)
$a \geq b$	b – a
$a > b$	b – a + k
A && B	max(cost(A), cost(B))
A || B	min(cost(A), cost(B))

Penghitungan nilai *cost* diperlukan untuk menentukan jumlah feromon yang akan ditinggalkan tiap semut di jalur yang terlewati. Jumlah feromon akan berpengaruh pada pemilihan jalur pada iterasi selanjutnya. Semakin Jumlah feromon yang ditinggalkan semut ditentukan oleh aturan *update* feromon. Pada penelitian ini, digunakan aturan *update* feromon yang telah dimodifikasi seperti pada penelitian Biswas et al.(2015). Berikut ini adalah aturan untuk *update* feromon :

$$\tau_{ij} = (1-\rho) * \tau_{ij} + \sum_{m}^{k=1} \Delta\tau_{ij}^{k}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{Q}{L_k} & jika\ semut\ k\ menggunakan\ edge\ (i,j) \\ 0 & sebaliknya \end{cases} \qquad \text{[III.2]}$$

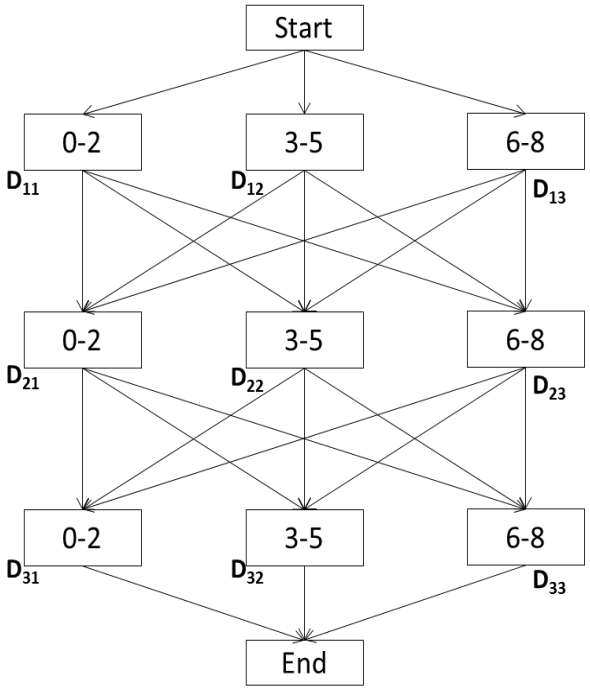
Keterangan :

- τ_{ij} : Jumlah feromon pada dari node i ke j
- ρ : Rasio penguapan feromon
- $\Delta\tau_{ij}^{k}$: Feromon per satuan panjang yang ditinggalkan semut ke-k
- L_k : Panjang jalur yang dilalui semut ke-k
- Q : konstanta

3.2.6 Demonstrasi pembangkitan data tes

Agar dapat menjelaskan langkah-langkah pembangkitan data tes dengan lebih rinci, dilakukan demonstrasi pembangkitan data tes untuk program Myers *triangle problem*. Hal pertama yang harus dilakukan adalah membuat model pencarian. Jumlah input pada program digambarkan sebagai *layer* pada *graph*. Program *myers triangle* memiliki jumlah input sebanyak 3 buah, maka model

pencariaanya terdiri dari 3 *layer*. Pada demonstrasi ini, bilangan input untuk sisi segitiga dibatasi dari 0-8 dan dikelompokkan ke dalam 3 domain. Sehingga, model pencariannya seperti gambar 3.9.



Gambar 3. 6 Model pencarian sisi segitiga

Langkah selanjutnya adalah memilih salah satu path dari CFG. Pada demonstrasi ini, path yang digunakan adalah path 1. Data tes yang dapat melewati path 1 harus benilai benar ketika melewati percabangan x<=0 || y<=0 || z<=0. Maka, nilai *costnya* adalah min((x-0),(y-0),(z-0)) sesuai dengan fungsi *cost* tabel 3.1. Nilai *cost* akan digunakan sebagai pengganti panjang lintasan (L_k) pada rumus *update* feromon. Karena *cost* dapat bernilai 0, maka pada penelitian ini L_k=cost+1 untuk menghindari *error* ketika bilangan dibagi dengan 0.

Jadi rumus untuk update feromon pada demonstrasi ini adalah :

$$\tau_{ij} = (1-\rho) * \tau_{ij} + \sum_{m}^{k=1} \Delta\tau_{ij}^{k}$$

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{1}{cost+1} & jika\ semut\ k\ menggunakan\ edge\ (i,j) \\ 0 & sebaliknya \end{cases} \qquad \text{[III.3]}$$

Setelah memilih path yang akan dicari data tesnya, langkah selanjutnya adalah menentukan feromon awal, rasio penguapan feromon, dan jumlah iterasi yang menggunakan strategi memilih yang terburuk (tempnum). Pada demonstrasi ini, feromon awal bernilai 1 dengan rasio penguapan (ρ) sebesar 0.1 dan tempnum sebanyak 3 iterasi. Selanjutnya, proses utama dari pembangkitan data tes dimulai. Pada awalnya sejumlah semut buatan diletakkan di *start node*. Pada demonstrasi ini, digunakan 5 ekor semut (k_1,k_2,k_3,k_4,k_5). Kemudian, setiap semut akan berjalan menuju *end note*. Ketika menentukan jalur selanjutnya, semut akan memilih jalur dengan feromon terendah di iterasi awal dan sebaliknya setelah iterasi awal. Jika terdpat dua atau lebih jalur dengan jumlah feromon sama, jalur dipilih secara acak.

Setelah semut-semut sampai di *end node*, dibangkitkan 3 bilangan sesuai dengan *node/domain* yang dilewati di setiap *layer* sebagai data tes sementara. Untuk mengetahui apakah data tes yang dihasilkan sudah sesuai dengan yang dibutuhkan, dilakukan penghitungan nilai *cost*. jumlah feromon pada tiap *edge* diperbarui dengan menggunakan rumus *update* feromon. Semakin besar nilai *cost*, jumlah feromon yang ditambahkan akan menjadi semakin sedikit. Setelah itu, semut-semut dikembalikan ke *start node* untuk iterasi selanjutnya.

Tabel 3. 2 Pemilihan jalur masing-masing semut

Iterasi	Semut	Path	Data tes			Cost
			x	y	z	
1	k_1	start > D_{13} > D_{21} > D_{33} > end	6	1	8	1
	k_2	start > D_{13} > D_{23} > D_{32} > end	7	6	4	4
	k_3	start > D_{13} > D_{21} > D_{33} > end	6	2	8	2
	k_4	start > D_{12} > D_{21} > D_{31} > end	5	1	2	1
	k_5	start > D_{13} > D_{23} > D_{31} > end	7	6	2	2
2	k_1	start > D_{11} > D_{23} > D_{33} > end	2	6	6	2
	k_2	start > D_{11} > D_{22} > D_{33} > end	1	4	7	1
	k_3	start > D_{11} > D_{21} > D_{32} > end	1	1	3	1
	k_4	start > D_{11} > D_{23} > D_{33} > end	1	7	8	1
	k_5	start > D_{11} > D_{22} > D_{33} > end	2	3	6	2
3	k_1	start > D_{12} > D_{22} > D_{31} > end	5	3	1	1
	k_2	start > D_{12} > D_{22} > D_{32} > end	4	3	4	3
	k_3	start > D_{12} > D_{23} > D_{32} > end	4	6	5	4
	k_4	start > D_{12} > D_{22} > D_{31} > end	3	3	1	1
	k_5	start > D_{12} > D_{22} > D_{32} > end	3	5	5	3
4	k_1	start > D_{12} > D_{22} > D_{31} > end	4	4	1	1
	k_2	start > D_{12} > D_{22} > D_{31} > end	5	3	2	2
	k_3	start > D_{12} > D_{22} > D_{31} > end	4	3	0	0

	k_4	start > D_{12} > D_{22} > D_{31} > end	3	3	1	1
	k_5	start > D_{12} > D_{22} > D_{31} > end	3	5	1	1

Tabel 3. 3 Jumlah feromon pada tiap edge setelah dilalui semut

Edge	Iterasi				
	0	1	2	3	4
Start > D_{11}	1	0.9	2.9767	2.679	2.4111
Start > D_{12}	1	1.4	1.26	2.834	5.3839
Start > D_{13}	1	2.2667	2.04	1.836	1.6524
D_{11} > D_{21}	1	0.9	1.31	1.179	1.0611
D_{11} > D_{22}	1	0.9	1.6433	1.479	1.3311
D_{11} > D_{23}	1	0.9	1.6433	1.479	1.3311
D_{12} > D_{21}	1	1.4	1.26	1.134	1.0206
D_{12} > D_{22}	1	0.9	0.81	2.229	4.8394
D_{12} > D_{23}	1	0.9	0.81	0.929	0.8361
D_{13} > D_{21}	1	1.7333	1.56	1.404	1.2636
D_{13} > D_{22}	1	0.9	0.81	0.729	0.6561
D_{13} > D_{23}	1	1.4333	1.29	1.161	1.0449
D_{21} > D_{31}	1	1.4	1.26	1.134	1.0206
D_{21} > D_{32}	1	0.9	0.81	0.729	0.6561
D_{21} > D_{33}	1	1.7333	2.06	1.854	1.6686
D_{22} > D_{31}	1	0.9	0.81	1.729	4.3894
D_{22} > D_{32}	1	0.9	0.81	1.229	1.1061
D_{22} > D_{33}	1	0.9	1.6433	1.479	1.3311

$D_{23} > D_{31}$	1	1.2333	1.11	0.999	0.8991
$D_{23} > D_{32}$	1	1.1	0.99	1.091	0.9819
$D_{23} > D_{33}$	1	0.9	1.6433	1.479	1.3311

Dari tabel 3.2 dan tabel 3.3, dapat dilihat bahwa pada iterasi ke-1 semut-semut berjalan secara acak karena belum ada perbedaan kadar feromon di tiap edge. Pada iterasi ke-2 dan ke-3, semut-semut akan memilih edge dengan dengan jumlah feromon terendah. Misalnya di terasi ke-2 semut yang berada di *start node* akan cenderung berjalan ke node D_{11}. Karena pada iterasi sebelumnya jumlah feromon di *edge* start-D_{11} lebih rendah dari start-D_{12} dan start-D_{13}. Setelah melewati 3 iterasi, aturan memilih yang terburuk diabaikan. Sehingga, pada iterasi ke-4, semut-semut memilih jalur dengan kadar feromon tertinggi yaitu start > D_{12} > D_{22} > D_{31} > end.

Pada akhinya, akan ada semut yang dapat menemukan data tes yang sesuai untuk *path* yang dipilih dari CFG. Data tes yang sesuai adalah yang memiliki nilai cost sebesar 0. Pada demonstrasi ini, data tes yang sesuai ditemukan di iterasi ke-4 oleh semut k_3. Jadi, data tes yang sesuai untuk path 1 pada program Myers *Triangle Problem* adalah 4,3, dan 0. Langkah selanjutnya adalah memilih path lain untuk kemudian dibangkitkan data tesnya dengan proses yang sama. Hingga pada akhirnya, akan ditemukan data tes yang sesuai untuk setiap path pada program.

3.3 Spesifikasi Sistem

Dalam penelitian ini, dibutuhkan beberapa *platform* yang digunakan untuk mengimplementasikan pembangkitan data tes. Berikut ini adalah *platform* yang digunakan selama penelitian :

1. *Software*
    - Windows 7
    - Netbeans IDE versi 8.2
2. *Hardware*
    - Laptop dengan Processor Intel Core i3 64 bit.

3.4 Hipotesis Penelitian

Dengan menggunakan program *benchmark* sebagai input, akan menghasilkan data tes untuk setiap path indepeden. Data tes yang dicari adalah yang fungsi ojektifnya bernilai 0. Data tes yang telah dihasilkan dapat digunakan untuk tahap selanjutnya pada pengujian pengujian perangkat lunak, yaitu pelaksanaan pengujian dan evaluasi hasil pengujian. Pada tahap pengujian, data tes yang telah dihasilkan dijadikan input untuk program benchmark. Kemudian pada tahap evaluasi, output dari setiap input dibandingkan dengan output yang diharapkan. Jika terdapat perbedaan, maka terdapat kesalahan pada program yang diuji.

BAB IV

HASIL DAN PEMBAHASAN

Pada bab ini akan dibahas mengenai implementasi dari setiap langkah yang digunakan untuk uji coba sistem. Kemudian, dijelaskan mengenai hasil dari uji coba tersebut dan pembahasannya. Selain itu, pada bab ini juga akan dipaparkan mengenai integrasi antara penelitian ini dengan kajian Al-qur'an dan Hadist.

4.1 Deskripsi Program

Aplikasi ini dibuat dengan tujuan untuk menghasilkan data tes secara otomatis dengan menggunakan *source code* program yang diuji sebagai input. Kemudian, aplikasi akan menguraikan *source code* berdasarkan pola percabangannya. Dari pola percabangan ini, akan dapat dibentuk *control flow graph* (CFG). Dari CFG tersebut, akan terbentuk path-path yang kemudian digunakan untuk membangkitkan data tes menggunakan algoritma *ant colony optimization* (ACO).

Data tes yang telah dihasilkan dapat digunakan untuk tahap selanjutnya pada pengujian pengujian perangkat lunak, yaitu pelaksanaan pengujian dan evaluasi hasil pengujian. Pada tahap pelaksanaan pengujian, data tes yang telah dihasilkan dijadikan input untuk program benchmark. Kemudian pada tahap evaluasi hasil pengujian, output dari setiap input dibandingkan dengan output yang diharapkan. Aplikasi ini dibuat menggunakan bahasa pemrograman Java dengan memanfaatkan *NetBeans* sebagai IDE (*Integrated Development Environment*).

4.2 Hasil

4.2.1 Data uji

Penelitian ini menggunakan benchmark data program publik yang juga digunakan oleh peneliti lain. Penggunaan program benchmark publik dimaksudkan agar hasil dari penelitian ini dapat divalidasi dengan hasil dari penelitian lain yang sejenis. Terdapat tiga program benchmark yang digunakan dalam penelitian ini yaitu *Myer's triangle, Michael's triangle,* dan *Sthamer's triangle.*

2. *Myers Triangle*

   Program ini menggolongkan segitiga atas dasar 3 sisi input sebagai bukan segitiga, segitiga sama kaki, segitiga sama sisi atau segitiga sembarang. *Control flow graph* yang dihasilkan memiliki 14 *node*.

2. *Michael Triangle*

   Program ini juga menggolongkan segitiga atas dasar 3 sisi input sebagai non-segitiga atau segitiga, yaitu, sama kaki, sama sisi atau sisi tak sama panjang. *Control flow graph* yang dihasilkan memiliki 26 *node*.

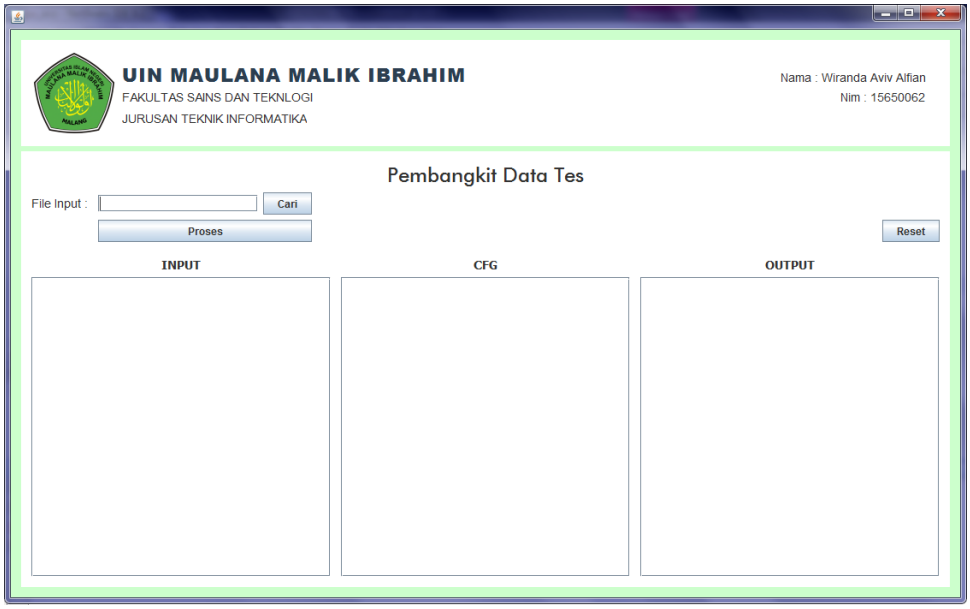
3. *Sthamer Triangle*

   Program ini juga menggolongkan segitiga atas dasar 3 sisi input sebagai bukan segitiga, segitiga, yaitu, sama kaki, sama sisi, segitiga siku-siku atau sisi tak sama panjang. *Control flow graph* yang dihasilkan memiliki 29 *node*.

4.2.2 Antarmuka aplikasi

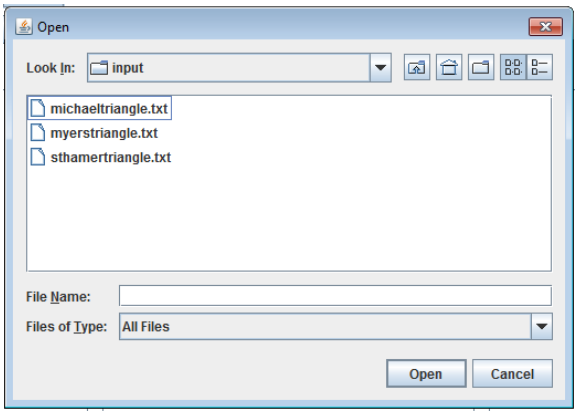
Setelah semua komponen pendukung telah siap maka langkah selanjutnya yaitu pembuatan program atau sistem informasi. Disini akan

dijelaskan fungsi-fungsi dan kegunaan program yang telah di buat beserta tampilan atau desainnya.



Gambar 4. 1 Desain antarmuka program

Gambar 4.1 merupakan tampilan halaman utama aplikasi. Terdapat tiga tombol pada halaman tersebut, yaitu tombol cari, proses, dan reset. Tombol cari berfungsi untuk membuka jendela pencarian file seperti pada gambar 4.2.



Gambar 4. 2 Jendela pencarian *file* input

File ini berisi *source code* yang akan dicari data tesnya. Tombol proses berfungsi untuk melakukan proses pembangkitan data tes dan menampilkan

hasilnya. tombol reset digunakan untuk mengosongkan kembali semua kotak input dan teks.

Selain itu, terdapat tiga kotak teks. Kotak input digunakan untuk menampilkan isi dari *source code* yang telah dipilih. Kotak CFG berfungsi untuk menampilkan hasil *parsing* dan kumpulan *graph* yang terbentuk.

4.2.3 Cara kerja aplikasi

Terdapat beberapa tahapan dalam pembuatan aplikasi pembangkitan data tes secara otomatis menggunakan algoritma *ant colony optimization* yang akan dibuat, berikut ini merupakan tahapan-tahapan proses yang ada dalam pembuatan aplikasi :

a. Parsing Program

Pada tahap *parsing* program, *source code* program yang telah diinputkan ke aplikasi akan dipecah-pecah setiap barisnya. *Parsing* dilakukan untuk menemukan pola percabangan yang terdapat di benchmark program. Pola dari percabangan tersebut nantinya akan digunakan dalam menyusun CFG.

b. Konversi ke CFG (Control Flow Graph)

Pada tahap ini, pola percabangan yang telah didapatkan melalui proses *parsing* akan diubah menjadi CFG. control flow graph (CFG) adalah representasi grafis dari aliran kontrol atau perhitungan selama eksekusi suatu program atau aplikasi. Control Flow Graph terdiri dari blok-blok yang berisi node mulai, node akhir, dan flow / arc diantara node-node tersebut.

Setelah itu, dari CFG yang telah terbetuk akan cari *path* independen linear. *Path* independen linear adalah *path* yang setidaknya

memiliki satu *node* baru. Tujuannya adalah untuk mengeksekusi semua jalur dan menemukan kemungkinan cacat dalam program.

c. Penerapan Metode ant colony optimization (ACO)

Untuk menghasilkan data tes dengan algoritma *ant colony*, program akan menyiapkan rasio penguapan, model pencarian, dan jumlah iterasi awal. Selama iterasi awal semut-semut memilih jalur dengan *cost* terbesar agar terhindar dari optimisasi lokal. Setelah itu untuk setiap *path* independen linear, akan dihasilkan data tes yang sesuai. Data tes yang sesuai adalah yang memiliki cost tidak lebih dari 0. Gambar 4.1 menunjukkan *source code* untuk menghasilkan data tes dengan menggunakan algoritma *ant colony optimization*.

```
public void generateDataTes(int path){
    int iter = 0;
    int cost = 99;
    int[] semut = {0,0,0};
    int[] dilewati={0,0,0};
    //int[] semut2 = {0,0,0};

    while (cost>0) {
        for(int layer=0;layer<3;layer++){
            int domain;
            if(iter<iterAwal){
                domain = terkecil(feromon[layer][0], feromon[layer][1], feromon[layer][2]);
            }else{
                domain = terbesar(feromon[layer][0], feromon[layer][1], feromon[layer][2]);
            }
            semut[layer] = pilihRandom(domain);
            dilewati[layer] = domain;
        }
        cost = fitness(semut,path);
        iterTotal++;

        System.out.println(semut[0]+" - "+semut[1]+" - "+semut[2]);
        System.out.println("cost : "+cost);
    }
    text_output.append(+semut[0]+"\t "+semut[1]+"\t "+semut[2]+"\n");
    resetFeromon();

}
```

Gambar 4. 3 *Source code* metode untuk menghasilkan data tes

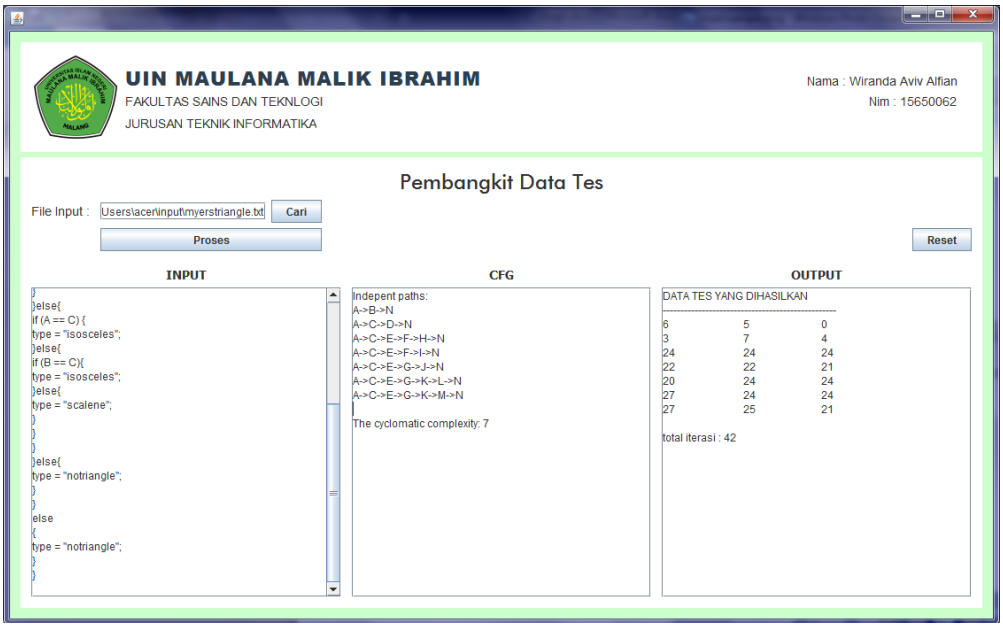
4.3 Pembahasan

Setelah pembuatan program untuk metode yang diusulkan sudah dilakukan, selanjutnya dilakukan uji coba progam. Pengujian akan dilakukan dengan memasukkan program *benchmark* sebagai *input*. Pengujian akan dilakukan

dengan masing-masing *benchmark* program yang digunakan. Penentuan parameter dan operator berdasarkan hasil beberapa pengaturan yang dilakukan oleh peneliti. Dari beberapa percobaan didapatkan beberapa pengaturan yang cukup memadai untuk digunakan di dalam pengujian.

4.3.1 Pengujian untuk program *benchmark Myers triangle*

Program Myers *triangle* adalah program yang menggolongkan segitiga atas dasar 3 sisi input. Kemudian, program akan menggolongkannya sebagai bukan segitiga, segitiga sama kaki, segitiga sama sisi atau segitiga sembarang.



Gambar 4. 4 Hasil analisa *Myers triangle*

Pada gambar 4.4, aplikasi menampilkan hasil analisa jarak *path* dan hasil akhir yang berupa data tes berdasarkan metode *ant colony optimization* (ACO). Kolom *Control Flow Graph* (CFG) menunjukkan pembagian node dari program *myers triangle* serta menampilkan *path-path* yang terbentuk. *Control Flow Graph* yang dihasilkan memiliki 14 *node* dengan *node* predikat berjumlah 7. Sedangkan jalur independen yang terbentuk sebanyak 7 *path*.

Berikut ini adalah jalur-jalur (*path*) yang dihasilkan dan akan dijadikan dasar dalam pembangkitan data tes:

A->B->N

A->C->D->N

A->C->E->F->H->N

A->C->E->F->I->N

A->C->E->G->J->N

A->C->E->G->K->L->N

A->C->E->G->K->M->N

Cyclomatic complexity: 7

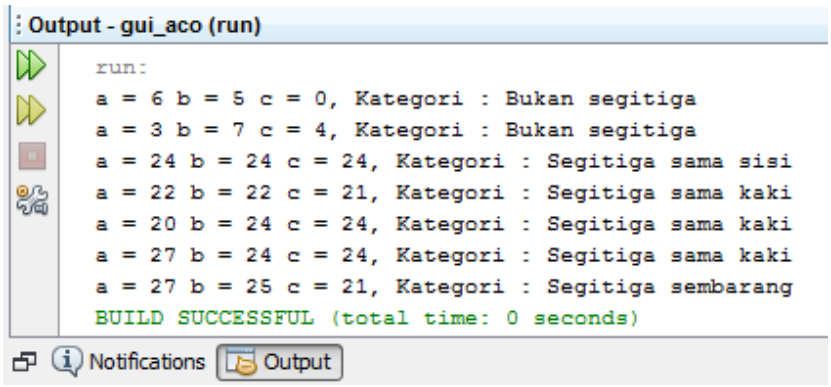
Data tes yang telah dihasilkan akan ditampilkan di kolom ouput. Data tes tersebut berupa 3 bilangan yang mewakili sisi-sisi segitiga. Data tes ini digunakan sebagai input untuk program benchmark myers triangle. Tabel 4.1 menunjukkan data tes yang dihasilkan sesuai dengan gambar 4.4

Tabel 4. 1 Data tes untuk program *Myers Triangle*

sisi 1	sisi 2	sisi 3
6	5	0
3	7	4
24	24	24
22	22	21
20	24	24
27	24	24
27	25	21

Tabel 4.1, menunjukkan data tes yang dihasilkan dengan menggunakan metode *ant colony optimization*. Untuk menegetahui apakah data tes yang

dihasilkan sudah sesuai, diperlukan validasi hasil. Proses validasi hasil dilakukan dengan cara menggunakan data tes yang telah dihasilkan sebagai *input* untuk menjalankan program *Myers triangle*.



Gambar 4. 5 Uji coba data tes pada program *benchmark Myers triangle*

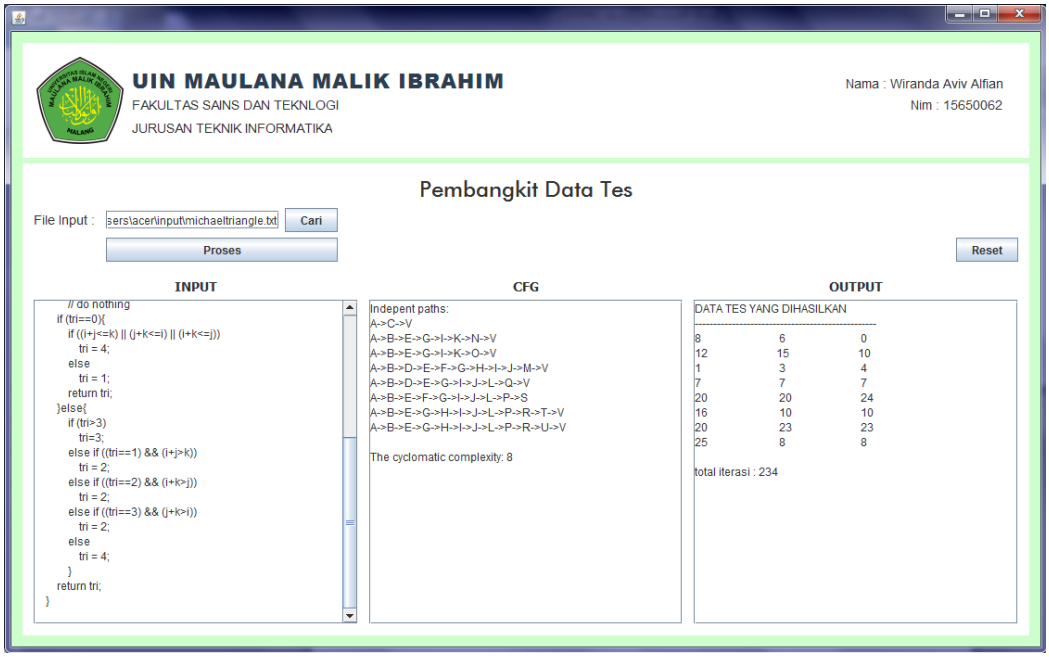
Gambar 4.5 Adalah *output* yang ditampilkan program *benchmark myers triangle* jika menggunakan data tes yang dihasilkan sebagai *input*. Ouput ini kemudian dibandingkan dengan *output* yang diharapkan sesuai dengan teori segitiga. Output yang ditampilkan sesuai dengan *output* yang diharapkan di tabel 4.2. Jadi, data tes yang dihasilkan telah sesuai.

Tabel 4. 2 Hasil validasi data tes dengan program *benchmark myers triangle*

Input			Output yang diharapkan	Output program *benchmark*	Status
a	b	c			
6	5	0	Bukan segitiga	Bukan segitiga	Sesuai
3	7	4	Bukan segitiga	Bukan segitiga	Sesuai
24	24	24	Segitiga sama sisi	Segitiga sama sisi	Sesuai
22	22	21	Segitiga sama kaki	Segitiga sama kaki	Sesuai
20	24	24	Segitiga sama kaki	Segitiga sama kaki	Sesuai
27	24	24	Segitiga sama kaki	Segitiga sama kaki	Sesuai
27	25	21	Segitiga sembarang	Segitiga sembarang	Sesuai

4.3.2 Pengujian untuk program benchmark Michael triangle

Program *michael triangle* adalah program yang menggolongkan segitiga atas dasar 3 sisi yang diinputkan. Kemudian, program akan menggolongkannya sebagai bukan segitiga, segitiga sama kaki, segitiga sama sisi atau segitiga sembarang.



Gambar 4. 6 Hasil analisa *Michael triangle*

Pada gambar 4.6, aplikasi menampilkan hasil analisa jarak *path* dan hasil akhir yang berupa data tes berdasarkan metode *ant colony optimization* (ACO). Kolom *Control Flow Graph* (CFG) menunjukkan pembagian node dari program *michael triangle* serta menampilkan *path-path* yang terbentuk. *Control Flow Graph* yang dihasilkan memiliki 22 *node* dengan *node* predikat berjumlah 6. Sedangkan jalur independen yang terbentuk sebanyak 8 *path*.

Berikut ini adalah jalur-jalur (*path*) yang dihasilkan dan akan dijadikan dasar dalam pembangkitan data tes :

A->C->V

A->B->E->G->I->K->N->V

A->B->E->G->I->K->O->V

A->B->D->E->F->G->H->I->J->M->V

A->B->D->E->G->I->J->L->Q->V

A->B->E->F->G->I->J->L->P->S

A->B->E->G->H->I->J->L->P->R->T->V

A->B->E->G->H->I->J->L->P->R->U->V

Cyclomatic complexity: 8

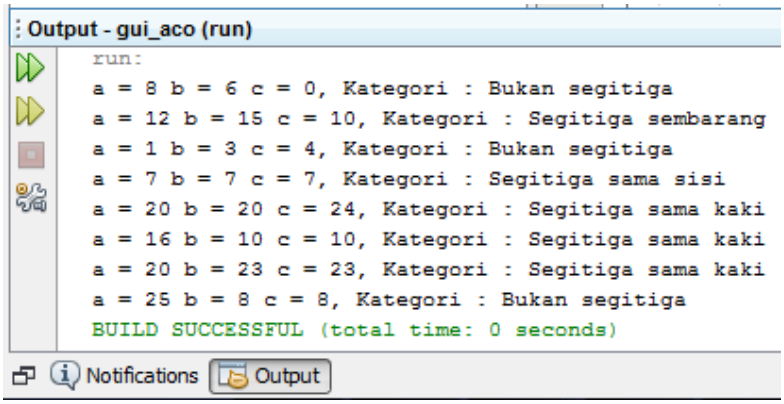
Data tes yang telah dihasilkan akan ditampilkan di kolom ouput. Data tes tersebut berupa 3 bilangan yang mewakili sisi-sisi segitiga. Data tes ini digunakan sebagai input untuk program benchmark michael triangle. Tabel 4.3 menunjukkan data tes yang dihasilkan sesuai dengan gambar 4.6.

Tabel 4. 3 Hasil data tes untuk program *Michael triangle*

sisi 1	sisi 2	sisi 3
8	6	0
12	15	10
1	3	4
7	7	7
20	20	24
16	10	10
20	23	23
25	8	8

Tabel 4.3 menunjukkan data tes yang dihasilkan dengan menggunakan metode *ant colony optimization.* Untuk menegetahui apakah data tes yang dihasilkan sudah sesuai, diperlukan validasi hasil. Proses validasi hasil

dilakukan dengan cara menggunakan data tes yang telah dihasilkan sebagai *input* untuk menjalankan program *michael triangle*.



Gambar 4. 7 Uji coba data tes pada program *benchmark michael triangle*

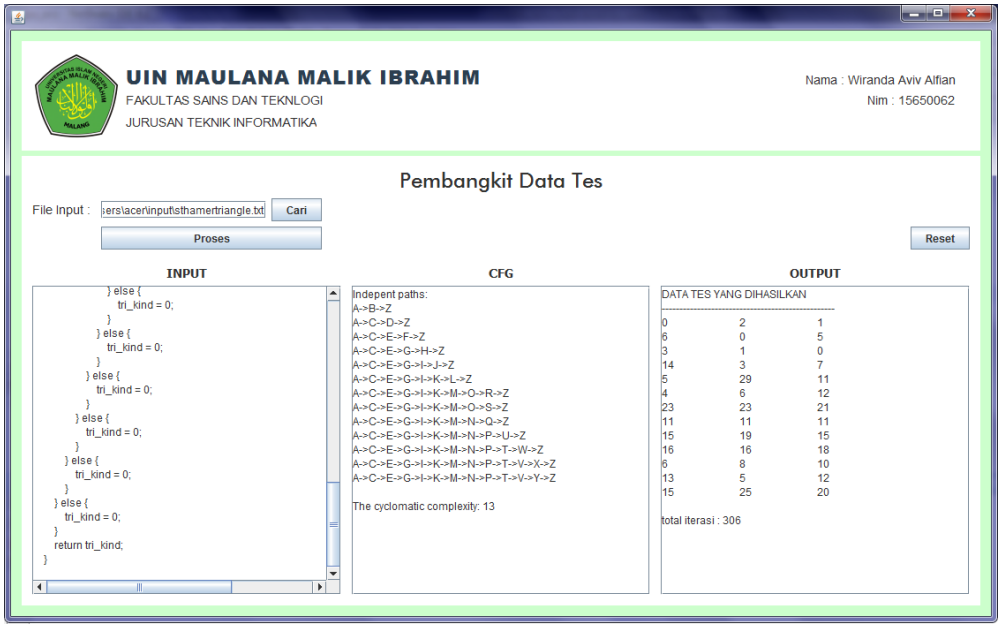
Gambar 4.7 Adalah *output* yang ditampilkan program *benchmark myers triangle* jika menggunakan data tes yang dihasilkan sebagai *input*. *Output* yang ditampilkan sama dengan *output* yang diharapkan di tabel 4.4. Kesimpulannya, data tes yang dihasilkan telah sesuai sebagai data tes untuk pengujian program *benchmark michael triangle*.

Tabel 4. 4 Hasil validasi data tes dengan program *benchmark michael triangle*

Input			Output yang diharapkan	Output program *benchmark*	Status
a	b	C			
8	6	0	Bukan segitiga	Bukan segitiga	Sesuai
12	15	10	Segitiga sembarang	Segitiga sembarang	Sesuai
1	3	4	Bukan segitiga	Bukan segitiga	Sesuai
7	7	7	Segitiga sama sisi	Segitiga sama sisi	Sesuai
20	20	24	Segitiga sama kaki	Segitiga sama kaki	Sesuai
16	10	10	Segitiga sama kaki	Segitiga sama kaki	Sesuai
20	23	23	Segitiga sama kaki	Segitiga sama kaki	Sesuai
25	8	8	Bukan segitiga	Bukan segitiga	Sesuai

4.3.3 Pengujian untuk program *benchmark Sthamer triangle*

Program *sthamer triangle* adalah program yang menggolongkan segitiga atas dasar 3 sisi yang diinputkan. Kemudian, program akan menggolongkannya sebagai bukan segitiga, segitiga sama kaki, segitiga sama sisi atau segitiga sembarang.



Gambar 4. 8 Hasil analisa *Sthamer triangle*

Pada gambar 4.8, aplikasi menampilkan hasil analisa jarak *path* dan hasil akhir yang berupa data tes berdasarkan metode *ant colony optimization* (ACO). Kolom *Control Flow Graph* (CFG) menunjukkan pembagian node dari program *sthamer triangle* serta menampilkan *path-path* yang terbentuk. *Control Flow Graph* yang dihasilkan memiliki 26 *node* dengan *node* predikat berjumlah 13. Sedangkan jalur independen yang terbentuk sebanyak 13 *path*.

Berikut ini adalah jalur-jalur (*path*) yang dihasilkan dan akan dijadikan dasar dalam pembangkitan data tes.

A->B->Z

A->C->D->Z

A->C->E->F->Z

A->C->E->G->H->Z

A->C->E->G->I->J->Z

A->C->E->G->I->K->L->Z

A->C->E->G->I->K->M->O->R->Z

A->C->E->G->I->K->M->O->S->Z

A->C->E->G->I->K->M->N->Q->Z

A->C->E->G->I->K->M->N->P->U->Z

A->C->E->G->I->K->M->N->P->T->W->Z

A->C->E->G->I->K->M->N->P->T->V->X->Z

A->C->E->G->I->K->M->N->P->T->V->Y->Z

Cyclomatic complexity: 13

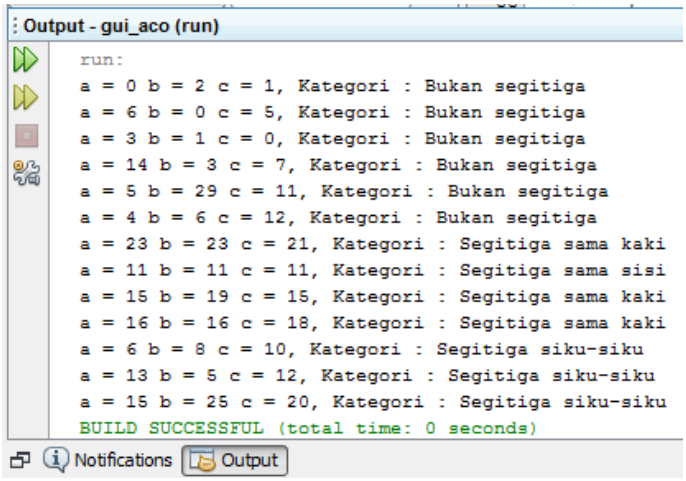
Data tes yang telah dihasilkan akan ditampilkan di kolom ouput. Data tes tersebut berupa 3 bilangan yang mewakili sisi-sisi segitiga. Data tes ini digunakan sebagai input untuk program benchmark sthamer triangle. Tabel 4.5 menunjukkan data tes yang dihasilkan sesuai dengan gambar 4.8.

Tabel 4. 5 Hasil data tes untuk program *Sthamer triangle*

sisi 1	sisi 2	sisi 3
0	2	1
6	0	5
3	1	0
14	3	7
5	29	11
4	6	12

23	23	21
11	11	11
15	19	15
16	16	18
6	8	10
13	5	12
15	25	20

Tabel 4.5 menunjukkan data tes yang dihasilkan dengan menggunakan metode *ant colony optimization*. Untuk menegetahui apakah data tes yang dihasilkan sudah sesuai, diperlukan validasi hasil. Proses validasi hasil dilakukan dengan cara menggunakan data tes yang telah dihasilkan sebagai *input* untuk menjalankan program *sthamer triangle*.



Gambar 4. 9 Uji coba data tes pada program *benchmark sthamer triangle*

Gambar 4.9 Adalah *output* yang ditampilkan program *benchmark sthamer triangle* jika menggunakan data tes yang dihasilkan sebagai *input*. *Output* yang ditampilkan sama dengan *output* yang diharapkan di tabel 4.6. Kesimpulannya, data tes yang dihasilkan telah sesuai sebagai data tes untuk pengujian program *benchmark sthamer triangle*.

Tabel 4. 6 Hasil validasi data tes dengan program benchmark michael triangle

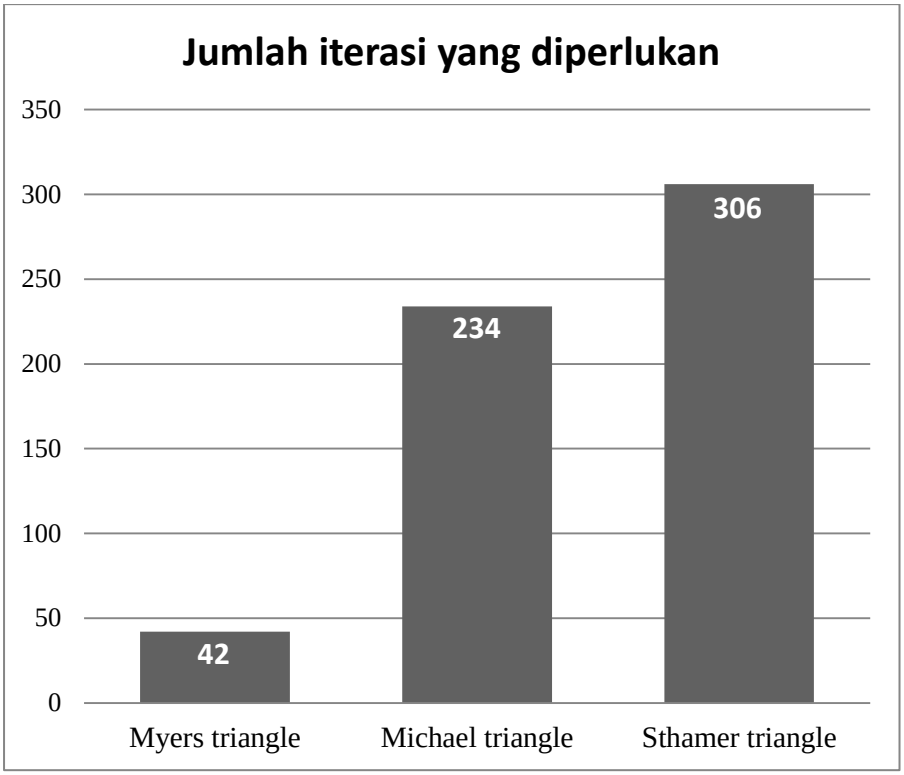
Input			Output yang diharapkan	Output program *benchmark*	Status
a	b	C			
0	2	1	Bukan segitiga	Bukan segitiga	Sesuai
6	0	5	Bukan segitiga	Bukan segitiga	Sesuai
3	1	0	Bukan segitiga	Bukan segitiga	Sesuai
14	3	7	Bukan segitiga	Bukan segitiga	Sesuai
5	29	11	Bukan segitiga	Bukan segitiga	Sesuai
4	6	12	Bukan segitiga	Bukan segitiga	Sesuai
23	21	21	Segitiga sama kaki	Segitiga sama kaki	Sesuai
11	11	11	Segitiga sama sisi	Segitiga sama sisi	Sesuai
15	19	15	Segitiga sama kaki	Segitiga sama kaki	Sesuai
16	16	18	Segitiga sama kaki	Segitiga sama kaki	Sesuai
6	8	10	Segitiga siku-siku	Segitiga siku-siku	Sesuai
13	5	12	Segitiga siku-siku	Segitiga siku-siku	Sesuai
15	25	20	Segitiga siku-siku	Segitiga siku-siku	Sesuai

4.4 Hasil Pengujian

Hasil pengujian dapat dilihat dari hasil validasi di tabel 4.2, tabel 4.4, dan tabel 4.6. *Output* dari program *benchmark* sesuai dengan *output* yang diharapkan. Jadi, data tes yang dihasilkan sudah sesuai dan dapat digunakan sebagai data tes untuk pengujian program.

Gambar 4.9 menunjukkan perbandingan iterasi yang diperlukan untuk masing-masing program *benchmark*. Masing-masing program *benchmark* memerlukan jumlah iterasi yang berbeda untuk menemukan data tes yang sesuai. Jumlah iterasi dipengaruhi oleh tingkat kompleksitas program yang diuji. Semakin tinggi tingkat kompleksitas suatu program, maka akan

memerlukan lebih banyak iterasi sehingga memerlukan waktu yang lebih lama untuk menghasilkan data tes.



Gambar 4. 10 Diagram perbandingan jumlah iterasi yang diperlukan untuk masing-masing program *benchmark*

4.5 Integrasi Sains dan Islam

Peran Al-Qur'an dalam pengembangan ilmu pengetahuan dan teknologi adalah sebagai paradigma atau pandangan dasar dalam mengembangkan ilmu pengetahuan. Dalam penelitian ini, ayat Al-Qur'an yang memiliki kaitan dengan pengujian perangkat lunak adalah surat Ali-Imran/3:186, yang berbunyi:

لَتُبْلَوُنَّ فِيٓ أَمْوَٰلِكُمْ وَأَنفُسِكُمْ وَلَتَسْمَعُنَّ مِنَ ٱلَّذِينَ أُوتُوا۟ ٱلْكِتَٰبَ مِن قَبْلِكُمْ وَمِنَ ٱلَّذِينَ أَشْرَكُوٓا۟ أَذًى

كَثِيرًا ۚ وَإِن تَصْبِرُوا۟ وَتَتَّقُوا۟ فَإِنَّ ذَٰلِكَ مِنْ عَزْمِ ٱلْأُمُورِ

Artinya: "Kamu sungguh-sungguh akan diuji terhadap hartamu dan dirimu. Dan (juga) kamu benar-benar akan mendengar dari orang-orang yang diberi al-Kitab sebelum kamu dan dari orang-orang yang

mempersekutukan Allah, gangguan yang banyak yang menyakitkan hati. Jika kamu bersabar dan bertakwa, maka sesungguhnya yang demikian itu termasuk urusan yang patut diutamakan." (QS. Ali-Imran/3 : 186).

Tafsir Ibnu Katsir menerangkan Firman Allah SWT yang artinya bhawa setiap manusia akan diuji terhadap harta dan dirinya. Ayat ini memiliki makna pada ayat lain yang berbunyi :

وَلَنَبْلُوَنَّكُمْ بِشَيْءٍ مِنَ الْخَوْفِ وَالْجُوعِ وَنَقْصٍ مِنَ الْأَمْوَالِ وَالْأَنْفُسِ وَالثَّمَرَاتِ ۗ وَبَشِّرِ الصَّابِرِينَ

Artinya : "Dan sungguh akan Kami berikan cobaan kepada kalian dengan sedikit ketakutan, kelaparan, kekurangan harta, jiwa, dan buah-buahan" (QS. Al-Baqarah/2: 155),

Dengan kata lain, seorang mukmin itu harus diuji terhadap sesuatu dari hartanya atau dirinya atau anaknya atau istrinya sesuai dengan tingkatan kadar agamanya, apabila agamanya kuat, maka ujiannya lebih dari yang lain.

Kemudian, Allah SWT juga berfirman kepada orang-orang mukmin ketika mereka tiba di Madinah sebelum Perang Badar untuk meringankan beban mereka dari tekanan gangguan yang menyakitkan hati yang dilakukan oleh kaum Ahli Kitab dan kaum musyrik. Sekaligus memerintahkan mereka agar bersikap pemaaf dan bersabar serta memberikan ampunan hingga Allah memberikan jalan keluar dari hal tersebut.

Selanjutnya, ayat tersebut memiliki makna bahwa Nabi dan para sahabatnya di masa lalu selalu bersikap pemaaf terhadap orang-orang musyrik dan Ahli Kitab, sesuai dengan perintah Allah kepada mereka, dan mereka bersabar dalam menghadapi gangguan yang menyakitkan. Tersebutlah bahwa Rasulullah bersikap pemaaf sesuai dengan pengertiannya dari apa yang diperintahkan oleh Allah

kepadanya, sehingga Allah mengizinkan kepada beliau terhadap mereka (yakni bertindak terhadap mereka).

Ujian hidup yang dialami setiap makhluk-Nya bertujuan untuk mengangkat derajat mereka. Sama halnya dengan tujuan dari pengujian perangkat lunak. Pengujian perangkat lunak bertujuan untuk meningkatkan kualitas perangkat lunak, dengan cara mencari celah dan kesalahan dari suatu perangkat lunak. Untuk kemudian dijadikan sebagai patokan perbaikan perangkat lunak.

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan pada hasil dari penelitian yang telah dilakukan, dapat disimpulkan bahwa :

1. Aplikasi yang telah dibuat berhasil menghasilkan data tes untuk program *benchmark*. Setelah dilakukan validasi terhadap data tes yang dihasilkan dengan menggunakan program *benchmark*, hasil yang dikeluarkan sesuai dengan output yang diharapkan.
2. Perbandingan waktu yang diperlukan untuk menghasilkan data tes dapat diketahui dari jumlah iterasi yang diperlukan. Jumlah ini bervariasi berdasarkan tingkat kompleksitas program yang diuji. Pada penelitian ini, progam *Sthamer's triangle* memerlukan paling banyak iterasi yaitu sebanyak 306. Kemudian, diikuti oleh program *Michael's triangle* sebanyak 234 iterasi dan *Myer's triangle* sebanyak 42 iterasi.

5.2 Saran

Peneliti menyadari bahwa penelitian ini memiliki banyak kekurangan dan diperlukan pengembangan lebih lanjut. Berikut ini adalah beberapa saran yang dapat dijadikan masukan untuk penelitian selanjutnya:

1. Pengujian dapat menggunakan atau mengkombinasikan dengan metode yang lain, agar menghasilkan data tes yang lebih baik.
2. Terdapat *bug* di aplikasi yang dibuat, dimana variable yang menyimpan *file* input tidak dapat dikosongkan kembali meskipun tombol reset sudah

ditekan. *Bug* ini dapat diatasi dengan membuka ulang aplikasi. Peneliti selanjutnya diharapkan dapat memperbaiki masalah ini.

DAFTAR PUSTAKA

Anand, S., et al. (2013). An Orchestrated Survey on Automated Software Test Case Generation. Generation, The Journal of Systems & Software

Biswas, S., Kaiser, M. S., & Mamun. S. A. (2015). Applying Ant Colony Optimization in Software Testing to Generate Prioritized Optimal Path and Test Data. International Conference ofn Electrical Engineering and Information Communication Technology (ICEEICT)

Ibn Katsir, Abu al-Fida' Ismail.(2014) Tafsir Ibnu Katsir. Terj. M. Abdul Ghoffar E.M `dan Abu Ihsan al-Atsari. Jilid VI. Cet. I. Bogor: Pustaka Imam asy-Syafi'i

Irwansyah, E., & Moniaga, J. V. (2014). Pengantar Teknologi Informasi. Deepublish.

Karjono, Moedjiono, & Kurniawan, D. (2016). Ant Colony Optimization. Jurnal TICOM Vol.4 No.3 Mei

Latiu, G. I., Cret, O. A., & Vacariu, L. (2012). Automatic Test Data Generation for Software Path Testing Using Evolutionary Algorithms. In 2012 Third International Conference on Emerging Intelligent Data and Web Technologies (pp. 1-8). IEEE.

Li, H., Lam, C. P. (2007). Software Test Data Generation using Ant Colony Optimization. International Journal of Computer, Electrical, Automation, Control and Information Engineering

Li, K., Zhang, Z., & Liu, W. (2009). Automatic Test Data Generation Based On Ant Colony Optimization. 2009 Fifth International Conference on Natural Computation

Mandyartha, E. P. (2017). Pembangkitan Data Uji Menggunakan Algoritma Genetika Multi-populasi Fuzzy Adaptif. Integer Journal

Manikumar, T., Kumar, A. J. S., & Maruthamuthu, R. (2016). Automated Test Data Generation for Branch Testing Using Incremental Genetic Algorithm. Sādhanā

Mao, C., Xiao, L., Yu, X., & Chen, J. (2014). Adapting Ant Colony Optimization to Generate Test Data for Software Structural Testing. Swarm and Evolutionary Computation

Maulana, M. R., Wahono, R. S., & Supriyanto, C. (2015). Integrasi Pareto Fitness, Multiple-Population dan Temporary Population pada Algoritma Genetika untuk Pembangkitan Data Test pada Pengujian Perangkat Lunak. Journal of Software Engineering

Michael, C. C., McGraw, G., and Schatz, M. (2001). Generating Software Test Data by Evolution. IEEE Trans. Softw. Eng. 27(12): 1085–1110

Myers, G. J., Sandler, C., and Badgett, T. (2011). The Art of Software Testing. John Wiley & Sons

Offut, A. Jefferson. dan Liu, Shaoying. (1999). Generating Test Data from SOFL Specifications. The Journal of Systems and Software.

Pachauri, A., Gursaran. (2012). Comparative Evaluation of a Maximization and Minimization Approach for Test Data Generation with Genetic Algorithm and Binary Particle Swarm Object. International Journal of Software Engineering & Application (IJSEA)

Pachauri, A., Srivastava, G. (2013). Automated Test Data Generation for Branch Testing Using Genetic Algorithm: An Improved Approach Using Branch Ordering, Memory and Elitism. The Journal of Systems and Software

Rerung, Rintho Rante. (2020). Algoritma dan Struktur Data untuk Perguruan Tinggi. Solok : Insan Cendekia Mandiri.

Rina, Violyta. (2009). Pengujian Perangkat Lunak. Fakultas Ilmu Komputer Universitas Indonesia.

Srivastava, P. R., Baby, K. (2010). Automated Software Testing Using Metahurestic Technique Based on An Ant Colony Optimization. International Symposium on Electronic System Design

Sthamer, H. H. (1995). The Automatic Generation of Software Test Data Using Genetic Algorithms. Doctoral dissertation, University of Glamorgan

Utting, M., & Legeard, B. (2007). *Pactical Model-based testing: A Tools Approach*. Morgan Kaufmann Publisher Inc.

Weisstein, Eric W. Triangle from MathWorld, A Wolfram Web Resource. https://mathworld.wolfram.com/Triangle.html diakses pada tanggal 21 Februari 2021

LAMPIRAN

1. Program *Myers Triangle*

```
public static void Triangle(double A,double B,
double C){
String type="";
double perimeter;
if (A > 0 && B > 0 && C > 0){
perimeter = A + B + C;
if (((2 * A) < perimeter) && ((2 * B) < perimeter) &&
((2 * C) < perimeter)){
if (A == B) {
if (B == C) {
type = "equilateral";
}else{
type = "isosceles";}
}else{
if (A == C) {
type = "isosceles";
}else{
if (B == C){
type = "isosceles";
}else{
type = "scalene";}
}}
}else{
type = "notriangle";}
}else
{
type = "notriangle";}
}
```

2. Program *Michael Triangle*

```
int triangle(double i, double j, double k){
int tri = 0;
if ((i<=0) || (j<=0) || (k<=0))
return 4;
else if (i==j)
tri += 1;
else
if (i==k)
tri += 2;
else
if (j==k)
tri += 3;
else
if (tri==0){
if ((i+j<=k) || (j+k<=i) || (i+k<=j))
tri = 4;
```

```
else
tri = 1;
return tri;
}else{
if (tri>3)
tri=3;
else if ((tri==1) && (i+j>k))
tri = 2;
else if ((tri==2) && (i+k>j))
tri = 2;
else if ((tri==3) && (j+k>i))
tri = 2;
else
tri = 4;}
return tri;}
```

3. Program *Sthamer Triangle*

```
static int triangle3(double a, double b, double c) {
double perim;
int tri_kind;
if (a > 0) {
if (b > 0) {
if (c > 0) {
perim = a + b + c;
if ((2 * a) < perim) {
if ((2 * b) < perim) {
if ((2 * c) < perim) {
if (a == b) {
if (b == c) {
tri_kind = 2;
} else {
tri_kind = 1;}
} else {
if (a == c) {
tri_kind = 1;
} else {
if (b == c) {
tri_kind = 1;
if ((a * a + b * b) == (c * c)) {
tri_kind = 3;
} else {
if ((b * b + c * c) == (a * a)) {
tri_kind = 3;
} else {
if ((a * a + c * c) == (b * b)) {
tri_kind = 3;
} else {
tri_kind = 4;}
}}}}
} else {
tri_kind = 0;}
} else {
```

```
tri_kind = 0;}
} else {
tri_kind = 0;}
} else {
tri_kind = 0;}
} else {
tri_kind = 0;}
} else {
tri_kind = 0;}
return tri_kind;
}
```