PENDETEKSIAN PLAGIARISME SOURCE CODE PADA PROGRAM BAHASA JAVA MENGGUNAKAN ALGORITMA TREE EDIT DISTANCE

SKRIPSI

Sebagai salah satu syarat untuk memperoleh gelar
Sarjana dalam bidang Ilmu Komputer

Oleh :

AFIN ANDIANTO

0610960003-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2011

UNIVERSITAS BRAWIJAYA

PENDETEKSIAN PLAGIARISME SOURCE CODE PADA PROGRAM BAHASA JAVA MENGGUNAKAN ALGORITMA TREE EDIT DISTANCE

SKRIPSI

Oleh :

AFIN ANDIANTO

0610960003-96

PROGRAM STUDI ILMU KOMPUTER

JURUSAN MATEMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS BRAWIJAYA

MALANG

2011

UNIVERSITAS BRAWIJAYA

LEMBAR PENGESAHAN SKRIPSI

PENDETEKSIAN PLAGIARISME SOURCE CODE PROGRAM BAHASA JAVA MENGGUNAKAN ALGORTIMA *TREE EDIT DISTANCE*

Oleh**:**
AFIN ANDIANTO
(0610960003-96)

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 6 januari 2011
dan dinyatakan memenuhi syarat untuk memperoleh
gelar Sarjana dalam bidang Ilmu Komputer

Pembimbing I	**Pembimbing II**
Drs. Achmad Ridok, M.Kom	**Dian Eka R, S.Si., M.Kom**
NIP. 196708011992031001	**NIP. 197306192002122001**

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya
Ketua,

Dr. Agus Suryanto, MSc
NIP. 19740414 200312 1 004

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama **:** Afin Andianto
NIM **:** 0610960003
Jurusan **:** Matematika
Penulis skripsi berjudul : Pendeteksian Plagiarisme *Source Code* Pada Program Bahasa Java Menggunakan Algoritma *Tree Edit Distance*.

Dengan ini menyatakan bahwa :

1. **Isi dari skripsi yang saya buat adalah benar-benar karya sendiri dan tidak menjiplak karya orang lain, selain nama-nama yang termaktub di isi dan tertulis di daftar pustaka dalam skripsi ini.**
2. **Apabila dikemudian hari ternyata skripsi yang saya tulis terbukti hasil jiplakan, maka saya akan bersedia menanggung segala resiko yang akan saya terima.**

Demikian pernyataan ini dibuat dengan segala kesadaran.

Malang, 6 Januari 2011
Yang menyatakan,

(Afin Andianto)
NIM. 0610960003

PENDETEKSIAN PLAGIARISME SOURCE CODE PADA PROGRAM BAHASA JAVA MENGGUNAKAN ALGORITMA TREE EDIT DISTANCE

ABSTRAK

Plagiarisme merupakan salah satu contoh dari budaya yang instan. Fenomena ini sudah semakin menjamur dalam berbagai bidang dan aspek kehidupan. Teknik plagiat terus berkembang dari waktu ke waktu karena didukung oleh kemajuan teknologi. Perkembangan teknik plagiat dapat dilihat dengan ditemukannya teknik plagiat mulai dari tingkat rendah sampai tingkat tinggi. Hal yang paling memprihatinkan adalah praktek plagiarisme yang dilakukan oleh kalangan akademis dalam bidang pendidikan. *Source code* merupakan salah satu bidang yang banyak ditemukan praktek plagiat. *Source code* memiliki elemen yang mudah untuk dimanipulasi sehingga tidak susah menemukan *source code* yang bersifat plagiat.

Pendeteksian plagiarisme *source code* java (*detecting similar java source code*) adalah proses menghitung kesamaan dua buah source code dengan memanfaatkan aplikasi yang dijalankan pada komputer. Kesamaan source code dihitung menggunakan algortima tree edit distance, yaitu dengan cara membentuk dan membandingkan tree dari elemen source code. Elemen source code diperoleh dari proses parsing dengan bantuan aplikasi javaparser, sedangkan perbandingan dilakukan dengan pendekatan metode *edit distance*. Pada penelitian ini dilakukan beberapa ujicoba terhadap source code. Hasil yang diperoleh dari ujicoba memiliki rata-rata diatas 90 % untuk source code yang memiliki kemiripan yang besar, dan menghasilkan rata-rata nilai sekitar 53,12 % untuk source code yang berberda.

DETECTING JAVA SOURCE CODE PLAGIARISM USING TREE EDIT DISTANCE ALGORITHM

ABSTRACT

Plagiarism is one example of the instant culture. This Phenomenon has been increasing in various field and aspect of life. Technique of plagiarism continue to grow over time as supported by technology. The development of plagiarism technique can be seen with the discovery of plagiarism technique from low level to high level.The most concern is the practice of plagiarism by academics in the field of education. Source code is one object in which many found the practice of plagiarism. Source code has element that is easy to manipulated so that is not difficult to find plagiarism source code.

Detecting java source code plagiarism is process of calculating the similarity of two source code, by utilizing application that run on the computer. The Similarity of source code are calculated using tree edit distance algortihm, in particular by establishing and comparing the tree from the element source code. Element Source code obtained from the parsing process with the help of javaparser application, while the comparison is done by edit distace algorithm. In this research, the application was conducted several test on the source code. Result obtained from the trial have averaged above 90 % for source code that has big simalrity, and generate an avaerage value of about 53,12 % for difference source code.

KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT, karena atas segala rahmat dan limpahan hidayah-Nya, penulis dapat belajar dan mengerjakan skripsi yang berjudul " Pendeteksian Plagiarisme *Source Code* Pada Program Bahasa Java Menggunakan Algoritma *Tree Edit Distance*".

Dalam penyelesaian skripsi ini, penulis telah mendapat begitu banyak bantuan baik moral maupun materiil dari banyak pihak. Atas bantuan yang telah diberikan, penulis ingin menyampaikan penghargaan dan ucapan terima kasih kepada:

1. Drs. Achmad Ridok, M.Kom., sebagai pembimbing I dan Dian Eka R, S.Si,. M.Kom., sebagai pembimbing II. Terima kasih atas waktu dan bimbingan yang telah diberikan.
2. Seluruh Bapak dan Ibu dosen yang telah mendidik dan memberikan ilmu kepada penulis.
3. Segenap staf dan karyawan di jurusan Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Brawijaya.
4. Ayah, Ibu, Adik dan seluruh keluarga. Terima kasih atas cinta, kasih sayang, doa dan semangat yang tiada henti.
5. Bagus Setiawan, Dedi Surahman, Risky Assegaf, Nandya Anantasari, Rikanda Ardiansyah, Lukman Hakim, M. Faizal Nugroho, Hermansyah Isfanhanani, dan Riezqy Rizal atas bantuan, dukungan, semangat dan doanya.
6. Teman-teman ilkomers angkatan 2006 dan seluruh warga Program Studi Ilmu komputer Universitas Brawijaya.
7. Pihak lain yang telah membantu dalam menyelesaikan penelitian ini yang tidak bisa penulis sebutkan satu persatu.

Penulis menyadari bahwa masih banyak kekurangan dalam laporan ini disebabkan oleh keterbatasan kemampuan. Oleh karena itu penulis sangat menghargai saran dan kritik yang sifatnya membangun demi perbaikan penulisan dan mutu isi penelitian ini untuk kelanjutan penelitian serupa di masa mendatang.

Malang, Januari 2011

Penulis

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR SOURCE CODE

BAB I
PENDAHULUAN

1.1 Latar Belakang

Jumlah kode program yang bersifat duplikat atau plagiat yang beredar didunia mencapai 20% bahkan lebih, dari seluruh jumlah kode program yang telah dibuat (Baker, 1995).Dunia pendidikan merupakan salah satu tempat, dimana banyak ditemukan praktek plagiat *source code*, karena lebih dari 50% siswa melakukan praktek plagiat *source code* dalam menyelesaikan tugas. (Culwin & Naylor, 2001). Plagiarisme kode program sering ditemui pada mata pelajaran pemrograman level rendah (Sheard, 2002).

Teknik plagiat terus mengalami perkembangan dari waktu ke waktu, sehingga belum ada satu metode atau algoritma yang selalu menghasilkan nilai optimal dalam mendeteksi plagiat. Plagiat *source code* berkembang dengan pesat karena jumlah *source code* yang beredar juga sangat banyak, sehingga peluang untuk melakukan tindakan plagiat *source code* sangat besar.

Beberapa teknik yang umum digunakan dalam praktek plagiat antara lain: merubah sususan pernyataan dalam dokumen, mengganti susunan kata yang membentuk kalimat dalam hal ini termasuk juga mengganti variabel dan konstanta, menambahkan pernyataan yang bersifat *redundant* (Ji J H,2007).Praktek plagiat pada *source code* dapat dilakukan dengan beberapa cara seperti : penambahan komentar, format penulisan yang dirubah, penggantian nama variabel, perubahan urutan algoritma yang tidak mengganggu jalannya program, prosedur dirubah menjadi fungsi atau sebaliknya, pemanggilan prosedur diganti isi prosedur itu sendiri (Willy G, Ronald A, Krisantus S,2005).

Penelitian tentang plagiat *source code* terus berkembang dari tahun ke tahun. Metode yang digunakan juga bermacam-macam seperti: *string matching*, *edit distance*, *brute force*, dan *boyer-moore*. Penelitian untuk mendeteksi plagiat *source code* tidak akan pernah berhenti karena setiap metode memiliki keterbatasan, dan tidak selalu menghasilkan nilai optimal pada semua kasus yang ditemukan.

Penelitian tentang plagiat *source code* menggunakan metode *tree*, pernah dilakukan oleh Tobias Sager, Abraham Bernstein, Martin Pinzger, Christoph Kiefer pada tahun 2006 menggunakan

algoritma *tree* sebagai parameter untuk mengetahui kesamaan diantara *source code*. Setiap *source code* akan di parse ke dalam bentuk *ASTNode*. Kemudian, *AST Node* yang terbentuk diubah ke dalam bentuk FAMIX. FAMIX yang terbentuk kemudian di ubah ke bentuk *tree*. Tingkat kesamaan diantara kedua *tree* yang terbentuk dihitung dengan menggunakan metode *edit distance*. Pada penelitian tersebut parameter yang menjadi dasar pembanding adalah perbedaaan bentuk *tree* satu dengan *tree* yang lainnya, namun tidak membandingkan isi setiap *node* dalam *tree*.

Sedangkan dalam penelitian yang berjudul "Pendeteksian Plagiarisme *Source code* Pada Program Bahasa Java Menggunakan *Tree Edit distance*" ini akan dibuat sistem pendeteksi kesamaan *source code* java dengan membandingkan isi setiap node antara *tree* satu dengan *tree* yang lainnya. *Node* pada penelitian ini berisi bentuk FAMIX dan diikuti oleh nama atau deklarasi pada *source code*. Perbandingan isi node antar *tree* akan dilakukan dengan menggunakan algoritma *edit distance*.

1.2 Rumusan Masalah

Berdasarkan kondisi pada latar belakang, maka dapat disusun rumusan masalah yaitu bagaimana membuat sistem yang mampu mendeteksi kesamaan *source code* pada kode program java dengan menggunakan algoritma *tree edit distance*. Serta menguji tingkat keberhasilan algoritma *tree edit distance* dalam mendeteksi kesamaan diantara *source code* pada kode program java.

1.3 Batasan Masalah

Adapun batasan-batasan masalah dalam pembuatan sistem pendeteksi kesamaan *source code* java adalah :

- *Source code* yang dibandingkan adalah kode program java J2SE.
- Sistem hanya membandingkan satu *source code* yang berisi satu *class* bukan berisi banyak *class*.
- *Source code* yang dibandingkan dianggap benar ketika di eksekusi (*compile*).

1.4 Tujuan

Tujuan dari penelitian ini adalah membuat sistem yang mampu mendeteksi kesamaan *source code* java dengan menggunakan algoritma *tree edit distance*, serta mengetahui tingkat keberhasilan sistem dalam mendeteksi kesamaan diantara *source code* java.

1.5 Manfaat

Manfaat yang dapat diperoleh dari penelitian ini anatara lain:

- Aplikasi yang dibuat dapat digunakan sebagai salah satu pertimbangan dalam menguji kesamaan diantara *source code*.
- Menguji seberapa efektif tingkat keberhasilan algoritma *tree edit distance* untuk mendeteksi kesamaan *Source code* pada kode program java.

1.6 Metodologi

Metode penyelesaian masalah yang dilakukan pada penelitian ini yaitu :

1. Studi Literatur

Membaca dan mempelajari beberapa literatur seperti jurnal, buku dan artikel dari website, yang berkaitan dengan plagiat *source code* dan algoritma *tree edit distance* yang dipakai untuk menguji kesamaan *source code* java.

2. Perancangan dan implementasi perangkat lunak.

Merancang dan membangun sebuah perangkat lunak berbasis java yang mengimplementasikan proses pendeteksi kesamaan *source code* program java menggunakan algoritma *tree edit distance*.

3. Uji coba dan analisis hasil implementasi

Menganalisis hasil implementasi, yaitu menguji tingkat kesamaan *code java* dari beberpa program java dengan beberapa modifikasi.

1.7 Sistematikan Penulisan

Pada laporan penulisan hasil penelitian ini menjabarkan keseluruhan kegiatan selama peneletian yang dikelompokkan secara sistematis menjadi lima bab. Adapun pembagian bab-bab tersebut adalah:

I. BAB I PENDAHULUAN

Dalam bab ini membahas mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi, serta sistematika penulisan.

II. BAB II TINJAUAN PUSTAKAN

Menjelaskan tentang dasar teori yang digunakan dalam menyusun laporan. Hal tersebut meliputi penjelasan definisi dari algoritma *tree edit distance*, pengertian java, rumusan pendeteksi kesamaan kode java.

III. BAB III METODE PERANCANGAN

Menjelaskan tentang rancangan sistem yang akan digunakan untuk mendeteksi kesamaan kode java. Rancangan tersebut meliputi metode metode yang akan digunakan beserta parameter-parameter yang dibutuhkan untuk proses pendeteksian.

IV. BAB IV IMPLEMENTASI DAN PEMBAHASAN

Pada bab ini akan dilakukan implementasi sistem, serta analisa hasil yang dikeluarkan oleh perangkat lunak, untuk selanjutnya dilakukan analisa hasil untuk mengevaluasi kinerja algoritma *tree edit distance* dalam mendeteksi kesamaan *code* java.

V. BAB V PENUTUP

Bab ini berisi tentang kesimpulan dari pembahasan bab sebelumnya serta saran dari penelitian yang telah dilakukan.

BAB II

TINJAUAN PUSTAKA

2.1 Pengertian Plagiat *Source code*

2.1.1 Pengertian Plagiat

Pengertian plagiat menurut kamus besar bahasa indonesia (KBBI) adalah pengambilan karangan (pendapat dsb) orang lain dan menjadikannya seolah-olah karangan (pendapat dsb) sendiri. Sedangkan pengertian plagiat menurut kamus webster adalah praktek mengklaim atau menyiratkan penulis asli (atau menggabungkan bahan dari) orang lain yang ditulis kreatif, secara keseluruhan atau sebagian, menjadi satu sendiri tanpa pengakuan yang memadai.

Tindakan yang dapat dianggap plagiat antara lain: mengambil karya orang lain, menyalin kata atau ide orang lain tanpa memberikan kompensasi, tidak memberikan tanda petik dalam kutipan yang diambil, memberikan informasi yang salah tentang sumber kutipan, mengubah kata-kata namun struktur kalimat tetap sama tanpa mencantumkan sumber (iParadigms teams,2009). Plagiat adalah teknik penyalinan atau meniru karya orang lain yang diklaim menjadi hasil karya sendiri. Tidak adanya motivasi maupun kemudahan dalam proses penyalinan dengan harapan tidak diketahui orang lain menjadi alasan utama terjadinya praktik plagiat. Beberapa jenis plagiat yang dikenal selama ini yaitu: menyalin setiap kata secara langsung tanpa diubah sedikitpun, menyalin dan menulis ulang karya tanpa merubah struktur kalimat, mengakui hasil karya orang lain sebagai hasil karya sendiri dengan menggantikan nama pengarang sebenarnya(Willy G, Ronald A, Krisantus S, 2005).

2.1.2 Pengertian *Source code*

Dalam ilmu komputer, *source code* (atau disebut juga *source*) adalah kumpulan pernyataan atau deklarasi bahasa pemrogramman komputer yang ditulis dan dapat di baca manusia. *Source code* memungkinkan programmer untuk berkomunikasi dengan komputer menggunakan beberapa perintah yang telah terdefinisi. *Source code* merupakan sebuah program yang biasanya dibuat dalam satu atau lebih *file* teks, kadang-kadang disimpan dalam database yang disimpan sebagai prosedur dan dapat juga muncul

sebagai potongan kode yang tercetak di buku atau media lainnya. Banyaknya koleksi *file source code* dapat diatur dalam direktori pohon, dalam hal ini mungkin juga dikenal sebagai *source tree*. Sebuah *source code* program komputer adalah kumpulan *file-file* yang diperlukan untuk mengkonversi dari manusia ke bentuk-dibaca beberapa jenis komputer-bentuk eksekusi. *Source code* mungkin akan diubah menjadi sebuah *file* eksekusi oleh kompilator, atau dijalankan secara langsung dari bentuk yang dapat di baca manusia dengan bantuan penterjemah. *Source code* dari program proyek besar adalah kumpulan semua *source code* dari semua program komputer yang membentuk proyek (ono w purbo,2009).

Menurut LINFO (*The Linux Information Project*) *source code* adalah versi original dari *software* yang ditulis orang dalam bentuk plain text (berbentuk alphabet dan numerik yang dapat dibaca manusia). Software mencakup sistem operasi, aplikasi program dan data yang didapat dijalankan oleh perangkat yang mengandung *micro processor* atau biasa disebut *processor* atau *Central Processing Unit* (CPU). *Source code* dapat ditulis sampai ribuan baris bahasa pemrograman, baberapa bahasa pemrograman yang populer meliputi C, C++, Java, PHP, Perl dan Phyton.

2.1.3 Pengertian Plagiat *Source code*

Plagiat *source code* adalah proses pengambilan *code* orang lain untuk digunakan dalam program sendiri tanpa mencantumkan identitas dan persetujuan dari pemilik *code*. Dalam program yang bersifat open *source* juga disertakan pernyataan atau *user aggrement* yang membatasi developer lain ketika memakai *code* program yang telah dibuat.

Menurut Willy G, Ronald A, dan Krisantus S, pada tahun 2005 praktek plagiat pada *source code* dapat dilakukan dengan beberapa cara seperti :

1. Penambahan komentar.
2. Format penulisan yang dirubah.
3. Penggantian nama variabel.
4. Perubahan urutan algoritma yang tidak mengganggu jalannya program.
5. Prosedur dirubah menjadi fungsi atau sebaliknya.
6. Pemanggilan prosedur diganti isi prosedur itu sendiri

2.2 Pengertian Java *Parser*

Java parser adalah sebuah metode yang digunakan untuk menelaah atau mem-*parsing* listing program java menjadi beberapa bagian, sehingga mudah untuk dimengerti dan dipelajari. Dalam penelitian ini java parser yang digunakan menggukan java 1.5 Parser yang dilengkapi AST (*Abstract Syntax Tree*) versi 1.0.8 yang diperoleh dari website http://*code*.google.com/p/javaparser/ . Java parser ini memiliki lisensi free sehingga siapapun boleh menggunakan dan memanfaatkan *tools* ini. Java parser ini bukan merupakan versi final namun masih dalam perkembangan ke arah yang lebih baik, oleh karena itu pada penelitian ini nanti banyak perubahan (*override*) metode yang dilakukan sesuai dengan kebutuhan, sehingga dapat dicapai hasil parser yang sempurna.

2.3 Pengertian FAMIX (*Famoos Informarion Exchange Model*)

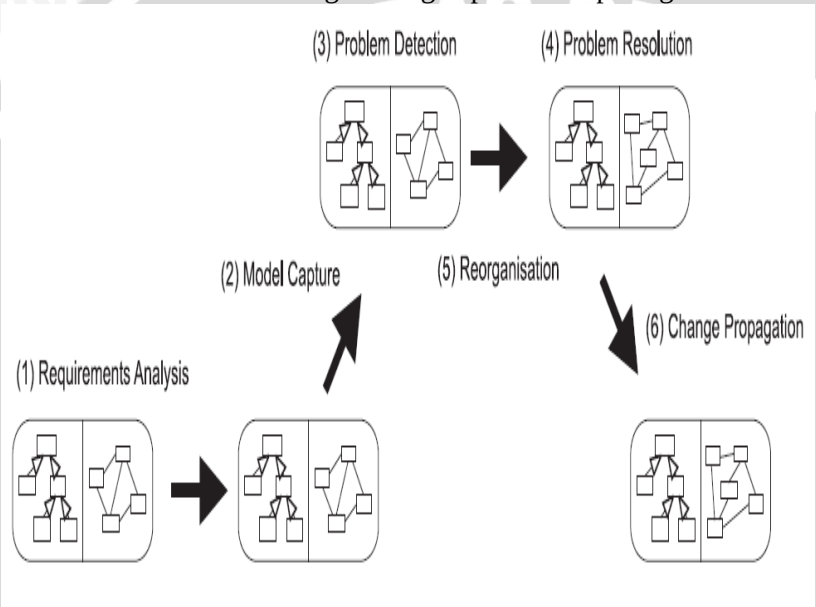
FAMIX pertama kali dikenalkan oleh Serge Demeyer, Sander Tichelaar dan Patrick Steyart. FAMIX merupakan format dari *object* oriented *source code* yang digunakan untuk bertukar informasi.FAMIX adalah sebuah bahasa pemrograman yang berdiri sendiri yang menggambarkan hubungan dari entitas yang dapat mendepenelitiankan *object-oriented code*. Format pertukaran informasi dalam FAMIX dikembangkan mulai tahun 1996 sampai dengan 1999(Petralli A C,2008).

Menurut Serge Demeyer, Sander Tichelaar dan Patrick Steyart tahun 1999, FAMIX dikembangkan dengan tujuan mengembangkan metode rekayasa ulang (*reengineering*) untuk mengubah kode program yang bersifat *object*-oriented kedalam sebuah *framework*. Metode rekayasa ulang tersebut meliputi:

1. *Requirement Analysis* : Mengidentifikasi tujuan dari proses *reengineering*.
2. *Model Capture* : Memahami dan mendokumentasikan sistem perangkat lunak.
3. *Problem Detection* : Mengidentifikasi masalah fleksibilitas dan kualitas.
4. *Problem Resolution* : Menyeleksi arsitektur program baru untuk menyelesaikan masalah yang telah diidentifikasi.

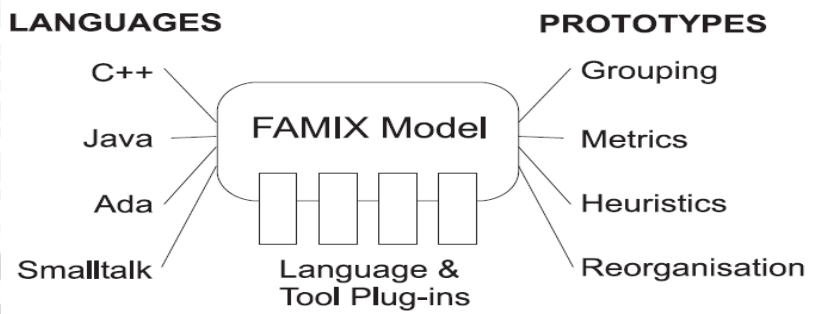
5. *Reorganisation* : Mengubah arsitektur software yang ada untuk rilis baru.
6. *Change Propagation* : Memastikan bahwa semua *client* dapat merasakan manfaat dari rilis baru yang telah diterbitkan.

Jika digambarkan dalam bentuk diagram, maka bentuk diagram dari metode FAMOOS reengineering dapat dilihat pada gambar 2.1.



Gambar : 2.1 FAMOOS *Reengineering Life Cycle*
(Serge Demeyer, Sander Tichelaar,Patrick Steyaert ,1999)

Secara konsep FAMIX berfungsi mengubah *object-oriented code* kedalam sebuah bentuk prototype yang sering digunakan sebagai model pertukaran informasi kode program untuk mencegah praktek plagiat. Gambar konsep dari FAMIX model dapat dilihat pada gambar 2.2.



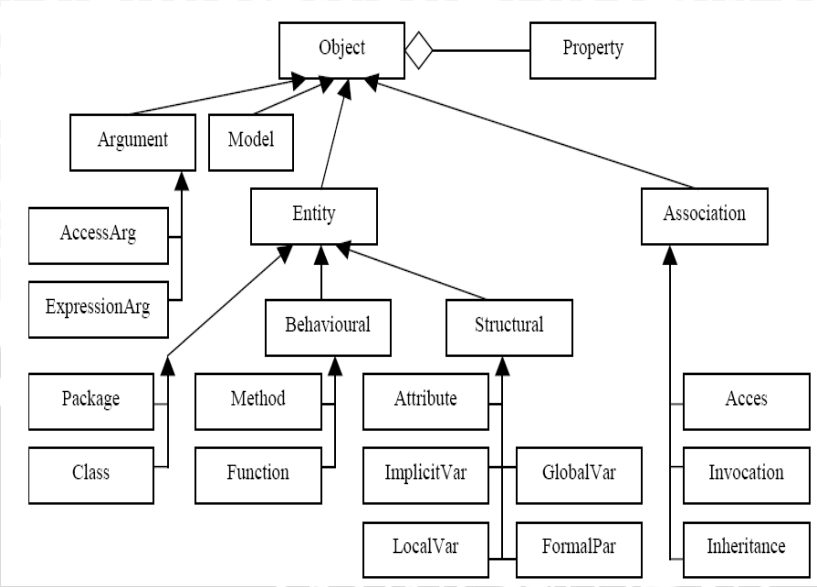
Gambar 2.2 Konsep FAMIX Model
(Serge Demeyer, Sander Tichelaar,Patrick Steyaert ,1999)

FAMIX terdiri dari beberapa entitas penting yang digunakan untuk membentuk model FAMIX dari sebuah *source code*. Entitas FAMIX dapat dilihat pada tabel 2.1.

Tabel 2.1 Entitas FAMIX

Entity	Sub*class* of
Package *Class*	Entitas Standar FAMIX
Method Function	Behavioral Entity (Entitas Perilaku)
Attribute ImpliciteVariable LocalVariable Globalvariable FormalParameter	Struktur Entitas
InhetitanceDefinition Invocation Access	Association Entity (Entitas Asosiasi)

Bentuk FAMIX menurut Michael Freidig 1999 pada jurnal yang berjudul XMI for FAMIX, dapat ditunjukkan pada gambar 2.3



Gambar 2.3 Diagram FAMIX Model
(Michael Freidig,1999)

Keterangan dari diagram depenelitian FAMIX pada gambar 2.3 adalah sebagai berikut :

- *Obejct* adalah *abstract class* tanpa *superclass*. Association dan Entitas adalah abstract *class* turunan dari *object*. Property adalah concrete *class* tanpa sebuah super*class*.
- Model merupakan informasi mengenai sistem tertentu yang sedang dimodelkan.
- *Package* merupakan nama dari sub-unit model *source code.* Contoh : namespace dalam C++ dan packages dalam Java.
- *Class* merupakan definisi nama kelas dalam sebuah *source code*.
- *BehaviouralEntity* menggambarkan tindakan atau perilaku yang dilakukan abstract *class*. Sub*class* dari *class* ini memiliki mekanisme yang berbeda untuk mendefinisikan entitas.
- *StructuralEntity* merupakan gambaran dari struktur entitas pada sebuah sistem. *StructuralEntity* kebanyakan berbeda

pada tiap waktu, namun ada juga yang tetap sama dari waktu ke waktu.

- *Attribute* menunjukkan state atau keadaan suatu kelas. Tergantung dari bahasa yang digunakan.
- *ImplicitVariable* adalah *key*word dalam *source code* yang mengacu pada lokasi memori pada saat memanggil *key*word teresebut. Pada Java dan C++ kita panggil dengan menggunakan *key*word **this**.
- *LocalVariable* merupakan entitas yang ditetapkan secara lokal ke dalam suatu tindakan atau perilaku.
- *FormalParameter* menggambarkan deklarasi variabel yang langsung dipanggil ketika pemanggilan suatu metode maupun fungsi.
- *InheritanceDefinition* menunjukkan hubungan inheritance atau turunan antara dua kelas. Diamna satu kelas sebagai super*class* dan yang lain sebagai sub*class*.
- *Access* menggambarkan bahwa entitas mengakses sebuah StructuralEntity. StructuralEntity dapat berupa attribut, local variabel, global variabel dan argumen.
- *Invocation* menggambarkan jika terdapat BehaviouralEntity memanggil *BehaviouralEntity* yang lain. Contoh memanggil *method* atau *function*. Jika *method* atau *function* yang sama dan dipanggil lebih dari satu kali, maka FAMIX model akan menghasilkan sebuah invocation.
- *Argument* menggambarkan berlalunya argumen saat menjalankan sebuah *BehaviouralEntity*. Model ini membedakan antara dua jenis argumen, yaitu *ExpressionArgument* atau *AccessArgument*.

Sebelum dibentuk kedalam format FAMIX *source code* melalui tahapan ektrasi untuk memperoleh informasi penting sebelum diubah ke bentuk FAMIX. Tahapan extraksi tersebut terbagi menjadi empat bagian yaitu :

- Level I : mencari nama *class*, definisi turunan *class*, *package*, *method* dan *fuction*.
- Level II : mencari attribut dan variabel global dari suatu *class*.

- Level III : menemukan entitas yang bersifat sebagai *invocation* maupun *access*.
- Level IV : menemukan entitas yang lain seperti argumen, variabel lokal, dan parameter yang digunan.

Berbagai Modifikasi FAMIX juga pernah dilakukan untuk mengoptimalkan pendeteksian surce *code*, seperti yang pernah dilakukan oleh Tobias Sager, Abraham Bernstein, Martin Pinzger, Christoph Kiefer yang dipublikasikan pada tahun 2006. Aturan tersebut dapat dilihat pada tabel 2.2

Tabel 2.2 Aturan pembentukan AST *Node* kedalam FAMIX

AST *Node*	FAMIX Element
Package Declaration	Package
Type Declaration	Class Inheritance Definition
Field Declaration	Attribute
Method Declaration	Method
Single Variable Declaration	Formal Parameter
Single Variable Declaration	Local Variable
Constructor Invocation Super Constructor Invocation Class Instance Creation Method Invocation Super Method Invocation	invocation
Field Access Super Field Access Simple Name Qualified Name	Access

Pada penelitian ini, proses konversi dari *source code* ke dalam bentuk FAMIX akan menggunakan metode seperti penelitian yang digunakan oleh Tobias Sager, , Abraham Bernstein, Martin Pinzger, Christoph Kiefer, seperti pada tabel 2.2

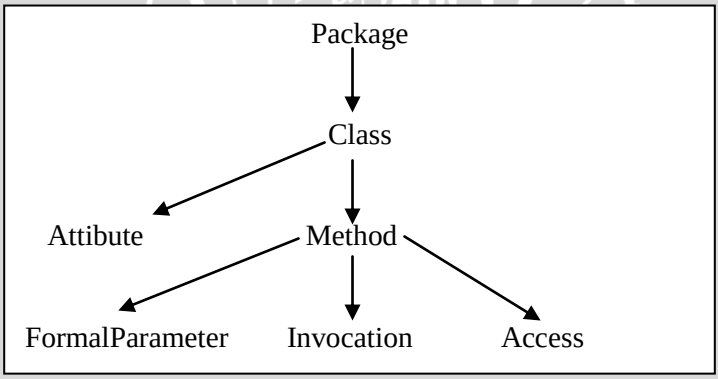
Contoh pembentukan bentuk FAMIX dari sebuah *source code* dapat dilihat pada *source code* 2.1 dan *source code* 2.2

Sample.java

```
Package test;
import java.awt.Color;
public class Sample extends Parent {
      private String space = " ";
      public Sample (int input) {
            System.out.println(Color.BLACK);
      }
}
```

Source code 2.1 Sample.java

Bentuk FAMIX dari *source code* 2.1 dapat dilihat pada gambar 2.4.



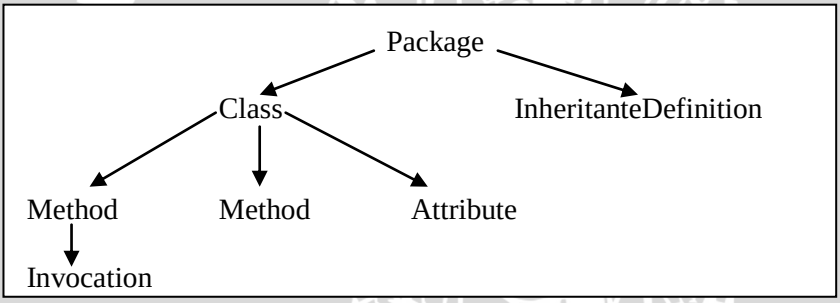
Gambar 2.4 Contoh Bentuk FAMIX dari Sample.java

Inheritance.java

```
package vehicles;
public class GeneralVehicle {
}
public class Car extends GeneralVehicle {
private String color = "";
public void setColor(String c) {
color = c;
}
private void recursiveMethod() {
recursiveMethod();
}
}
```

Source code 2.2 Inheritance.java

Bentuk FAMIX dari *source code* 2.2 dapat dilihat pada gambar 2.5.

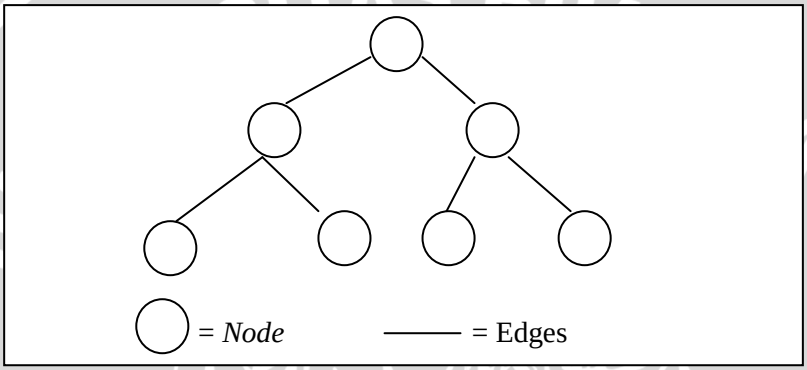


Gambar 2.5 Bentuk FAMIX dari Interitance class

2.4 **Algoritma *Tree Edit distance***

2.4.1 Algoritma *Tree*

Tree adalah sebuah algoritma yang menggambarkan bentuk alur ke dalam representasi pohon. *Tree* terdiri dari *node* dan *edges*, *node* digambarkan dengan simbol lingkaran, sedang *edges* adalah garis yang menggabungkan *node*. Gambar *tree* dapat dilihat pada gambar 2.6.



Gambar 2.6 *Tree*.

Elemen – elemen yang terdapat dalam *tree* menurut Robet Lafore dalam bukunya yang berjudul "*Data Structures and Algorithms in Java*" yang diterbitkan pada tahun 1998, antara lain :

1. *Path* merupakan lintasan yang menghubungkan tiap *node*.
2. *Root* adalah *node* yang paling atas dari sebuah *tree*.
3. *Parent* adalah *node* yang berada satu tingkat diatas, semua *node* pasti memiliki parent kecuali root.
4. *Child* adalah *node* yang berada satu tingkat dibawah.
5. *Leaf* adalah *node* yang tidak memiliki *child*. Satu *tree* hanya memiliki satu *root*, tapi bisa memiliki banyak *leaf*.
6. *Subtree*, setiap *node* bisa dianggap sebagi *root* dari child atau *node* dibawahnya. Sehingga *subtree* bisa lebih dari satu untuk setiap *tree*.
7. *Visiting* adalah tindakan yang melihat atau mebandingkan *node* yang telah ada sebelum menentukan lokasi *node* yang baru.

8. *Traversing* adalah *visit* semua *node* yang memiliki tujuan tertentu, seperti *sorting*.
9. *Level* tiap *node* menunjukkan berapa generasi *node* tersebut. Level tertinggi adalah *root*, sedangkan level terendah adalah *leaf*.
10. *Key*, setiap *node* memiliki *key* yang berbeda antara satu dengan yang lainnya. *Key* ini digunakan untuk mengenali *node* pada sebuah *tree*.

2.4.2 Algoritma *Edit distance*

Edit distance atau levenshtein *edit distance* adalah metode menghitung perbedaan antar kata dengan mengukur tingkat perubahan yang diperlukan dari kata satu ke bentuk kata yang dijadikan pembanding. Algoritma ini akan menghasilkan nilai yang besar apabila kata yang dibandingkan memiliki perbedaan yang besar. Semakin besar nilai *edit distance* yang dihasilkan maka perbedaan antar kata juga semakin besar. Algoritma ini pertama kali ditemukan oleh Vladimir Levenshtein pada tahun 1965(Michael Gilleland,2008).

Algoritma *Edit distance* adalah algoritma yang menghitung jarak atau perbedaan antara satu *object* dengan *object* yang lain. Dalam contoh kasus pembedaan diantara kata, *edit distance* merupakan besarnya perubahan terkecil yang mungkin dilakukan untuk merubah dari bentuk satu kata ke kata yang lain. *Edit distance* merupakan algoritma yang dapat digunakan untuk mengetahui tingkat kemiripan antara suatu *object* dengan *object* yang lainnya (Jun-ichi Aoe,1994).

Contoh penggunaan algoritma *edit distance* dalam kasus kalimat dapat dilihat pada gambar 2.7 dan 2.9. Proses perhitungan dapat dilihat pada gambar 2.8 dan gambar 2.10.

```
S1 = G  U  M  B  O
     |     |  |  |
S2 = G  A  M  B  O  L
d(S1,S2)  = 0 + 1+  0+  0+ 0+ 1  = 2
```

Gambar 2.7 Contoh penerapan algoritma *Edit distance*

Proses perhitungan *edit distance* pada gambar 2.7 terdapat perbedaan 2 karakter diantara S1 dan S2 sehingga nilai *edit distance* yang dihasilkan adalah 2. Proses perhitungan nilail *edit distance* pada gambar 2.7 dapat dilihat pada gambar 2.8.

		G	A	M	B	O	L
	0	1	2	3	4	5	6
G	1	0	1	2	3	4	5
U	2	1	1	2	3	4	5
M	3	2	2	1	2	3	4
B	4	3	3	2	1	2	3
O	5	4	4	3	2	1	2

Gambar 2.8 Contoh proses perhitungan *Edit distance*

```
S1 = r  u  m  a  h     s  a  k  i  t
     |  |  |                    |  |
S2 = r  u  m                    i  t
d(s1,s2)  = 0+0+0+1+1+1+1+1+1+0+ 0 = 6
```

Gambar 2.9 Contoh penerapan algoritma *Edit distance2*

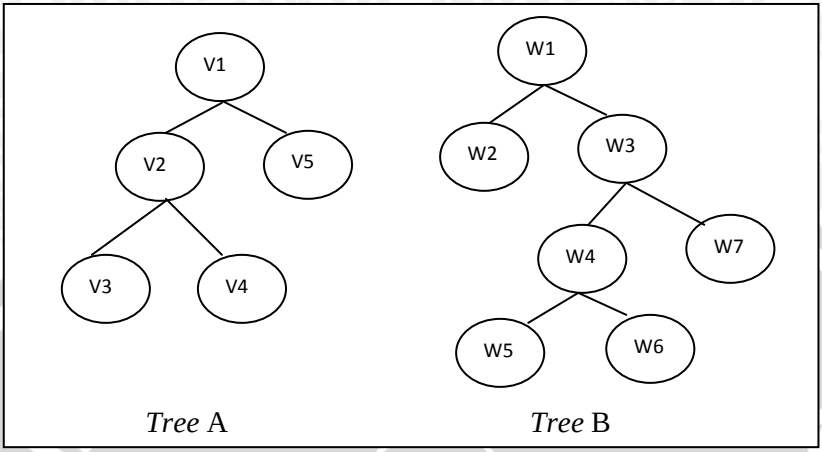
Pada proses perhitungan *edit distance* gambar 2.9 karakter ‘i’ dan ‘t’ pada S2 mengalami proses pergeseran ke dalam posisi yang sama dengan karakter ‘i’ dan ‘t’ pada S1 hal ini dikarenakan *edit distance* menghitung nilai paling minimal antara S1 dan S2. Jika tidak terjadi pergeseran maka nilai *edit distance* yang dihasilkan menjadi 8. Proses perhitungan nilail *edit distance* pada gambar 2.9 dapat dilihat pada gambar 2.10.

		R	U	M	A	H		S	A	K	I	T
	0	1	2	3	4	5	6	7	8	9	10	11
R	1	0	1	2	3	4	5	6	7	8	9	10
U	2	1	0	1	2	3	4	5	6	7	8	9
M	3	2	1	0	1	2	3	4	5	6	7	8
I	4	3	2	1	0	1	2	3	4	5	6	7
T	5	4	3	2	1	0	1	2	3	4	5	6

Gambar 2.10 Contoh proses perhitungan *Edit distance2*

2.4.3 Algoritma *Tree Edit distance*

Algortima *tree edit distance* adalah algortima yang menghitung perbedaan antara satu *tree* dengan *tree* yang lain. Algortima ini menghitung jumlah perbedaan diantara *tree* dengan mengapliksikan operasi – operasi dasar dalam pembuatan sebuah *tree*. Operasi – operasi tersebut meliputi penyisipan (*insertion*), pergantian (*substitution*) dan penghapusan (*deletion*) *node*. Contoh gambaran *edit distance* dapat diilustrasikan seperti gambar 2.6.



Gambar 2.11 Ilustrasi *Tree Edit distance Algorithm*
(Tobias S, Abraham B, Martin P, Christoph K, 2006)

Ilustrasi algoritma *Tree Edit distance* pada gambar 2.11 :

1. Root *tree* A = V1 dan root *tree* B = W1.
   Pada level ini tidak terdapat perbedaan sehingga dapat dinotasikan dengan (v1, w1).
2. *Node* v2 memiliki posisi sama dengan w2, sehingga dapat dinotasikan dengan (v2,w2).
3. *Node* v5 memiliki posisi sama dengan w3, sehingga dapat dinotasikan dengan (v5,w3).
4. *Node* v3 pada *tree* A tidak memiliki kesamaan dengan *node* apappun pada *tree* B, sehingga notasi diganti note pada *tree* B dengan simbol λ. Oleh karena itu notasi menjadi (v3, λ).
5. Sama dengan *node* v3, *node* v4 juga tidak memiliki kesamaan apapun pada *tree* B. Sehingga dapat dinotasikan dengan (v4, λ).
6. *Node* w4, w5, w6, w7 pada *tree* B tidak memiliki kesamaan apapun pada *tree* A. Sehingga notasi untuk ke empat *node* tersebut adalah (λ,w4), (λ,w5), (λ,w6) dan (λ,w7).
7. Notasi lengkap dari transformasi *tree* A dan *tree* B dapat dituliskan sebagai berikut [(v1, w1), (v2,w2), (v3, λ), (v4, λ), (v5,w3), (λ,w4), (λ,w5), (λ,w6) , (λ,w7)]

Untuk nilai *TreeDistance* diantara *node*, akan bernilai 1 apabila salah satu dari notasi diantara *tree* dinotasikasikan dengan simbol λ. Apabila pada notasi tidak ditemukan simbol λ maka nilai *TreeDistance* bernilai 0. Tingkat perbedaan diantara *tree*, jika menggunakan algoritma *tree edit distance* dapat dihitung dengan menggunakan persamaan 2.1

$$TreeEditDistance(T1,T2) = \frac{TreeDistance(T1,T2)}{|V1| + |V2|} \quad \textbf{(2.1)}$$

Persentase kesamaan diantara *tree* dapat dihitung menggunkan persamaan 2.2.

$$SimTreeDistance(T1,T2) = \frac{(|V1| + |V2|) - TreeEditDistance(T1,T2)}{|V1| + |V2|} x\ 100\% \quad \textbf{(2.2)}$$

Dimana, $|V1|$ = Jumlah node pada *tree* 1
$|V2|$ = Jumlah node pada *tree* 2
$TreeEditDistance(T1,T2)$ = Nilai perbedaan antara *Tree* 1 dan *Tree* 2.
(Tobias S, Abraham B, Martin P, Christoph K, 2006)

Jika diterapkan pada ilustrasi pada gambar 2.11 maka langkah perhitungan similaritas T1 dan T2 adalah sebagai berikut :

1. Hitung *TreeDistance*

   Tree Distance (T1,T2) = (v1, w1), (v2,w2), (v3, λ), (v4, λ), (v5,w3), (λ,w4), (λ,w5), (λ,w6) , (λ,w7)
   = 0+0+1+1+0+1+1+1+1
   = 6.

2. Hitung |V1| + |V2|

   $|V1| + |V2| = JumlahNodeTreeI + JumlahnodeTree2$
   = 5 + 7 = 12

3. Sim*TreeDistance*(T1,T2)

$$SimTreeDistance(T1,T2) = \frac{(|V1| + |V2|) - TreeEditDistance(T1,T2)}{|V1| + |V2|} x100\%$$

= ((5+7) – 6) /12 = 0,5 x 100% = 50%

Nilai similaritas antara *tree* 1 dengan *tree* 2 sebesar **50%.**

Pada tahap membandingkan isi *node* pada *tree*, yang dibandingkan adalah *string* yang ada disetiap *node* dengan syarat bahwa *node* yang akan dibandingkan memiliki kedudukan (*path*) yang sama pada tiap *tree*. Sedangakan *node* yang tidak memiliki kesamaan *path* dibandingkan dengan *string* kosong. Hal itu dikarenakan tiap *path* memiliki bentuk FAMIX yang berbeda. Proses perbandingan *string* dalam node menggunakan algortitma *edit distance*. Persamaan untuk mencari perbedaaan *string* tiap node menggunakan *edit distance* dapat dilihat dari persamaan 2.3.

$$EditDistance(T1,T2) = \frac{\sum(d(T1,T2))}{getMaks(|T1|,|T2|)} \quad \textbf{(2.3)}$$

Dimana, $\sum(d(T1,T2))$ = Jumlah total perbedaan perbandingangn *string* diantara node pada *Tree* 1 dan *Tree* 2.

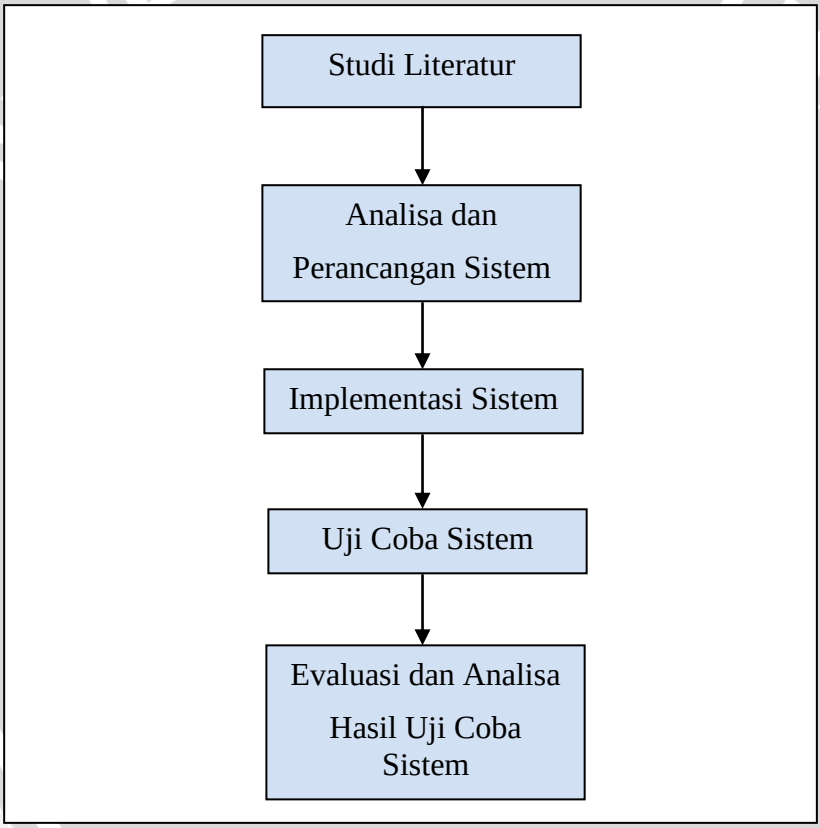
$getMaks(|T1|,|T2|)$ = Jumlah *string* terpanjang semua node diantara *Tree I* dan *Tree 2*.

Persamaan untuk similaritas *string* dengan menggunakan algoritma *edit distance* dapat dilihat pada persamaan 2.4.

$$SimEditDistance(T1,T2) = \frac{1}{1 + EditDistance\ (T1,T2)} x100\% \quad \textbf{(2.4)}$$

BAB III
METODE PERANCANGAN

Sistem ini dirancang untuk dapat menghasilkan aplikasi yang mampu mendeteksi kesamaan antara *source code* bahasa pemrograman java dengan menggunakan algoritma *tree editdistance*. Sistem ini akan diterapakan pada mesin komputer personal (PC) atau notebook dengan aplikasi berbasis java. Pada sistem ini akan diujikan beberapa jenis *source code* java dengan beberapa souce code java yang berbeda. Gambaran proses penelitian yang akan dilakukan dapat dilihat pada gambar 3.1



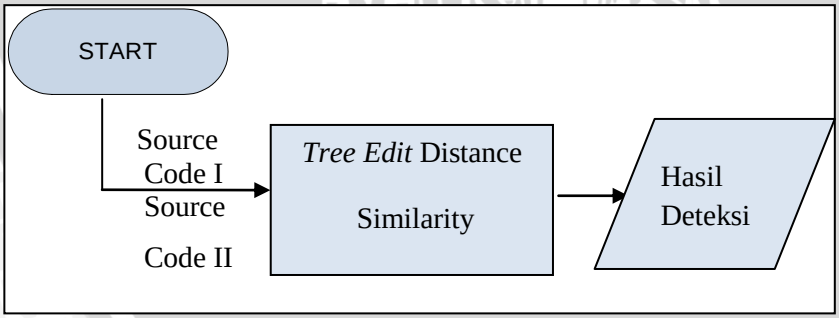
Gambar 3.1 Proses penelitian

Adapun tahapan yang dilalui dalam pembuaatan sistem ini adalah sebagai berikut :

1. Studi Literatur. Melakukan studi literatur terhadap algorima *tree editdistance*.
2. Analisa dan Perancangan Sistem. Menganalisa dan merancang perangakat lunak untuk menerapkan algoritma *tree edit distance*.
3. Implementasi Sistem. Mengimplementasikan perangkat lunak berdasarkan analisa dan perancangan yang telah dibuat.
4. Uji Coba Sistem. Melakukan uji coba terhadap perangkat lunak dan mengumpulkan hasil pengujian yang telah dilakukan.
5. Evaluasi dan Analisa Sistem. Melakukan evaluasi terhadap perangkat lunak dan data hasil uji coba yang diperoleh.

3.1 Perancangan sistem secara keseluruhan

Secara umum, sistem akan memiliki fungsi untuk menerima input dua buah *source code* dari *user*, dari dua buah *Source code* inputan, akan diketahui tingkat kesamaan atau kemiripan antara kedua *source code* tersebut dengan menggunakan metode *tree editdistance*. Sehingga *user* dapat mengetahui tingkat kemiripan diantara dua *source code* tersbut. Jika digambarkan dalam skema aliran data maka kinerja sistem akan nampak seperti pada gambar 3.2.



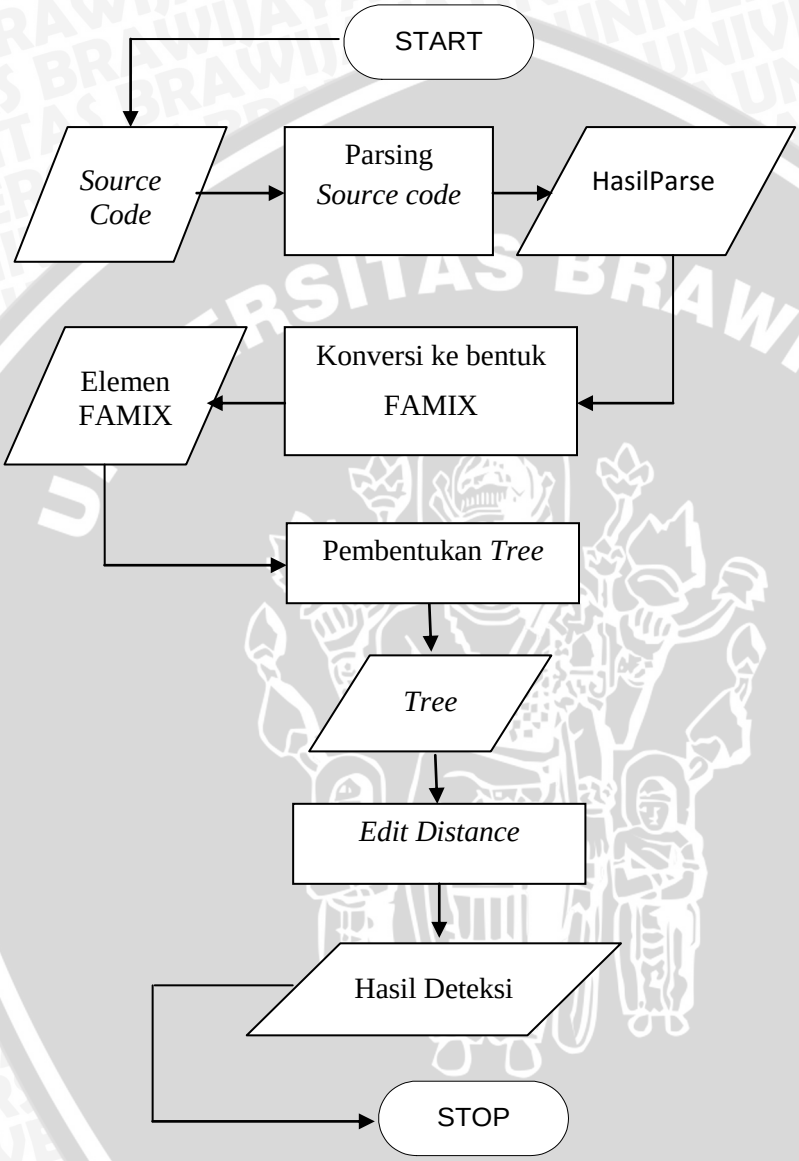
Gambar 3.2 Aliran Kinerja Sistem

3.2 Perancangan Proses

Perangkat lunak yang akan dibuat adalah aplikasi untuk mendeteksi kesamaan code java secara otomatis menggunakan algoritma *tree editdistance*. *Source code* dalam hal ini merupakan *input* atau masukkan dari *user* berupa *file* dalam bentuk listing program java yang berekstnsi *.java* . Kemudian user memasukkan lagi *source code* yang telah dimodifikasi, maka sistem akan mulai melakukan proses pendeteksian kesamaan diantara kedua *source code* tersebut.

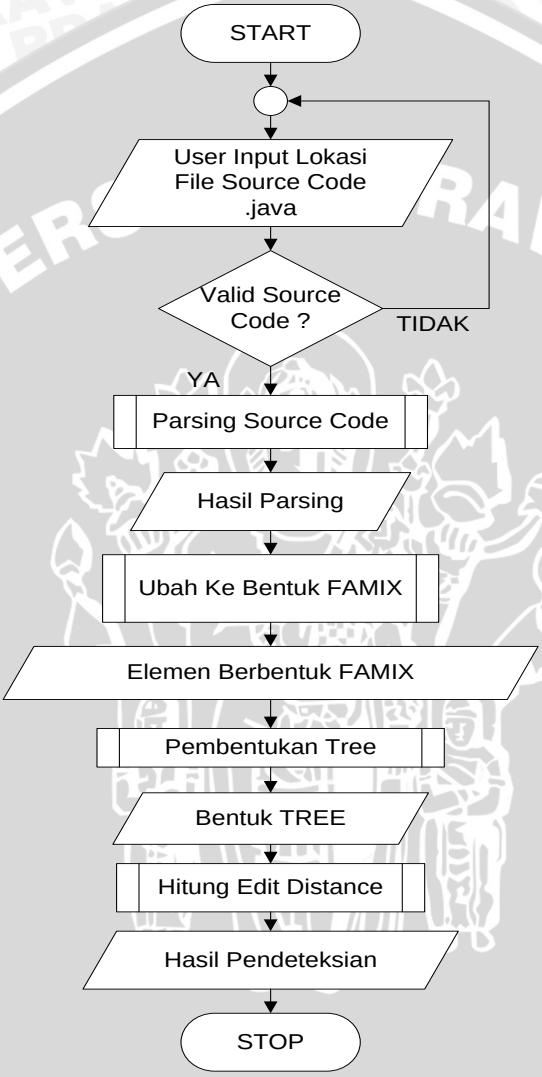
Proses pendeteksian kesamaan *source code* yang dilakukan sistem diawali dengan membaca dan mengekstrak informasi *source code* yang dilakukan oleh sistem, seperti nama file, ukuran file dan lokasi tempat penyimpanan file. Setelah proses pengambilan informasi dilakukan, maka langkah kedua adalah *mem-parsing source code* untuk diketahui c*lass name*, *package*, *method* atau *function*, *atributte*, *constuctor* dan *variable* pada masing – masing *source code*. Setelah diketahui komponen yang membentuk *source code*, kemudian dilakukan proses pengubahan komponen ke dalam bentuk FAMIX 2.0. Dalam proses ini komponen pembentuk *source code* akan diganti nama dengan inisialisasi pada FAMIX 2.0 dengan demikian, maka komponen *source code* sudah menjadi bentuk baku dan siap diubah ke bentuk *tree*.

Jika *tree* dari bentuk FAMIX pada masing-masing telah terbentuk, maka tahap berikutnya adalah membandingkan isi *node* antara *tree* dari *source code* orisinil dengan *tree* yang telah mengalami modifikasi pada *source code* nya, dengan menggunakan algoritma *editdistance*. Proses langkah kerja sitem pendeteksi kesamaan antara *source code* dapat dilihat pada gambar 3.3.



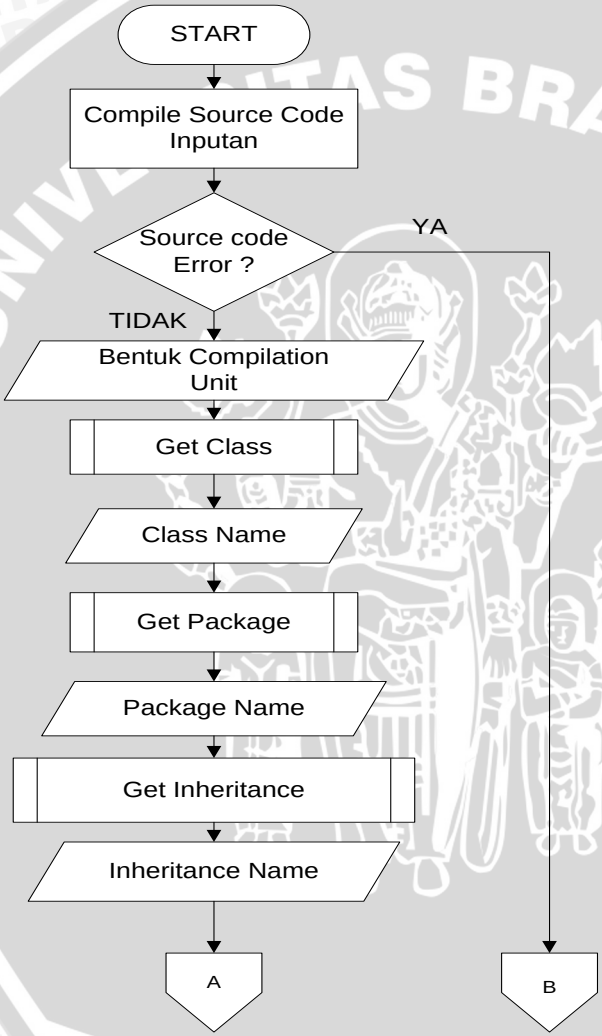
Gambar 3.3 Flowchat Langkah Kerja Sistem

Flowchart sistem pendeteksi kesamaan *source code* pada bahasa pemrograman java dapat dilihat pada gambar 3.4.

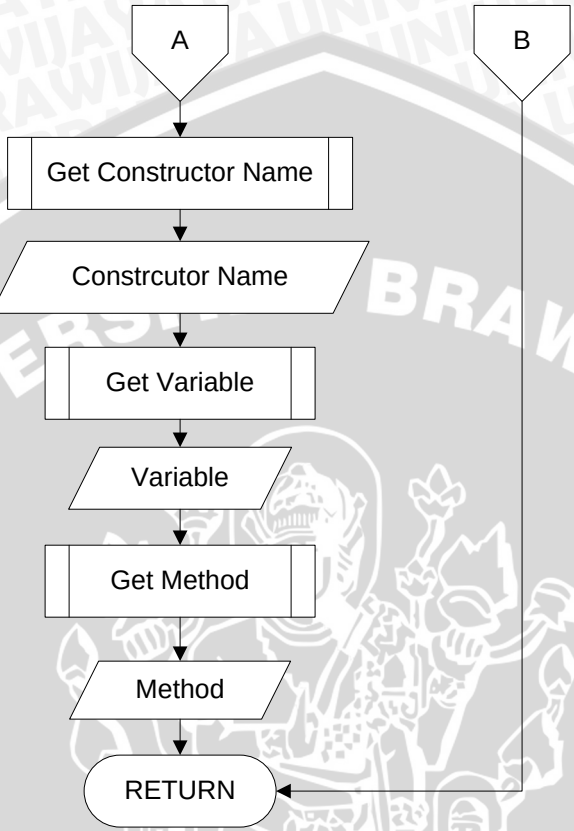


Gambar 3.4 Flowchart Proses Sistem

Proses parsing *source code* pada penelitian ini menggunakan java parser yang dikembangkan oleh jgesser, untuk mendapatkan elemen pembentuk *tree*. Elemen tersebut meliputi nama file, variabel yang digunakan, methode dan fungsi yang dipakai. Perancangan proses parsing source code dapat dilihat pada gambar 3.5 dan gambar 3.6.

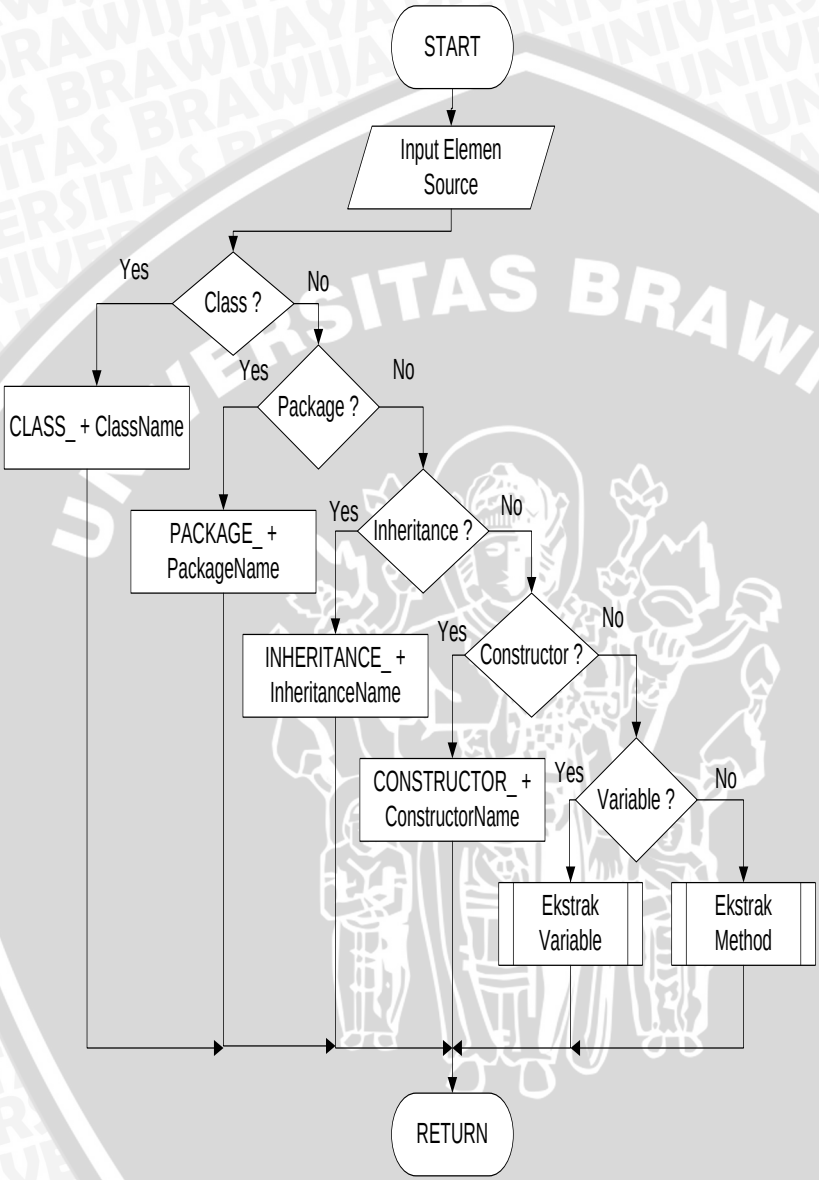


Gambar 3.5 Flowchart parse source code I

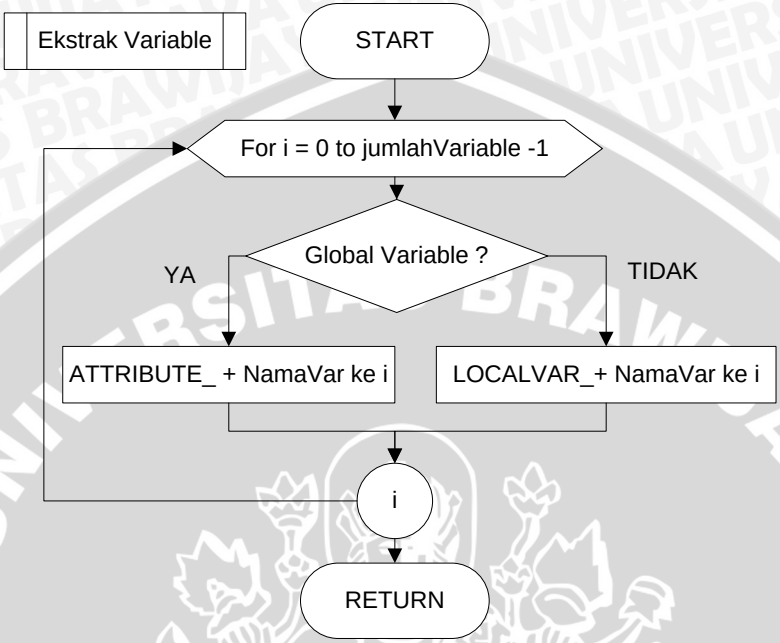


Gambar 3.6 Flowchart parse source code II

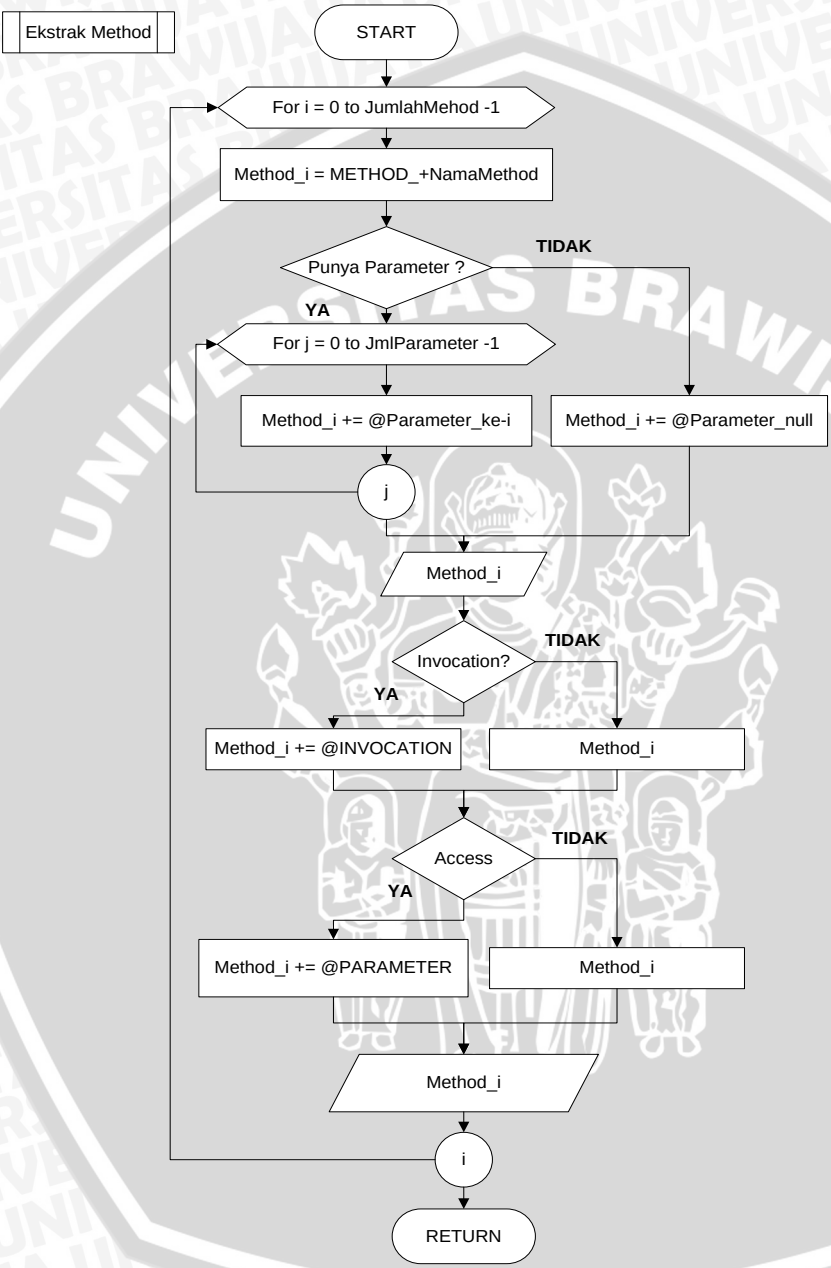
Proses parsing tidak dijelaskan dengan detail karena proses parsing menggunakan tools javaparser yang diperoleh dari website http://code.google.com/p/javaparser/. Proses pembentukan elemen source code kedalam bentuk FAMIX dapat dilihat pada gambar 3.7, 3.8 dan 3.9.



Gambar 3.7 Flowchart perubahan elemen source code ke FAMIX I

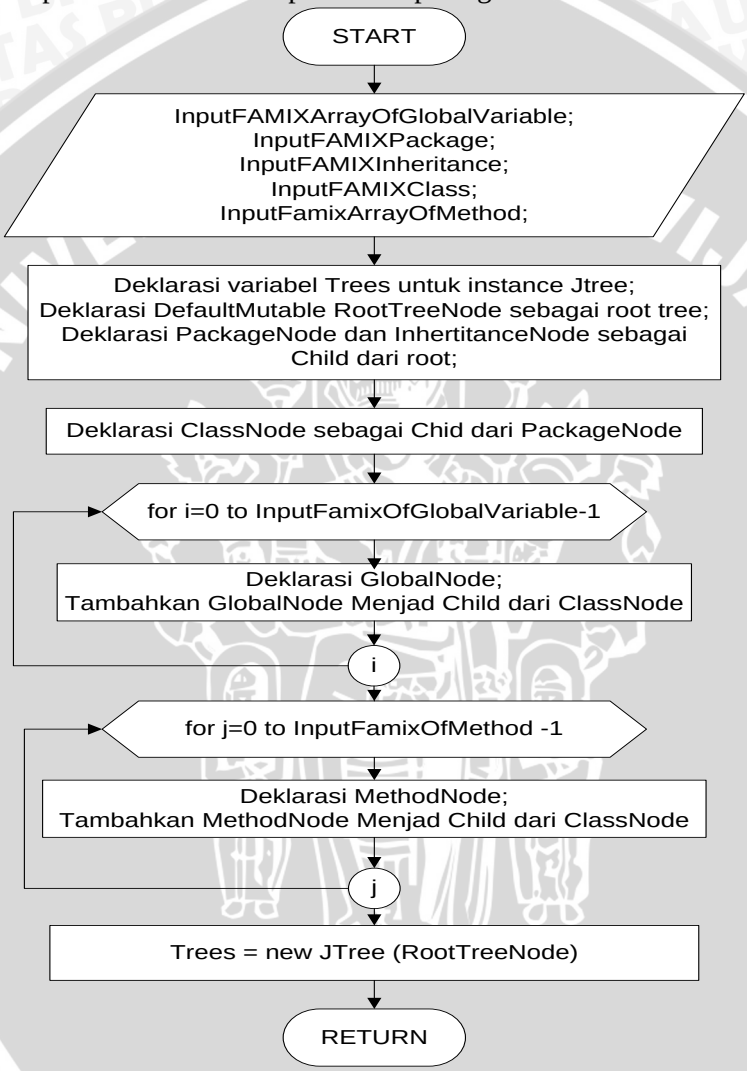


Gambar 3.8 Flowchart Ekstrak *Variable*



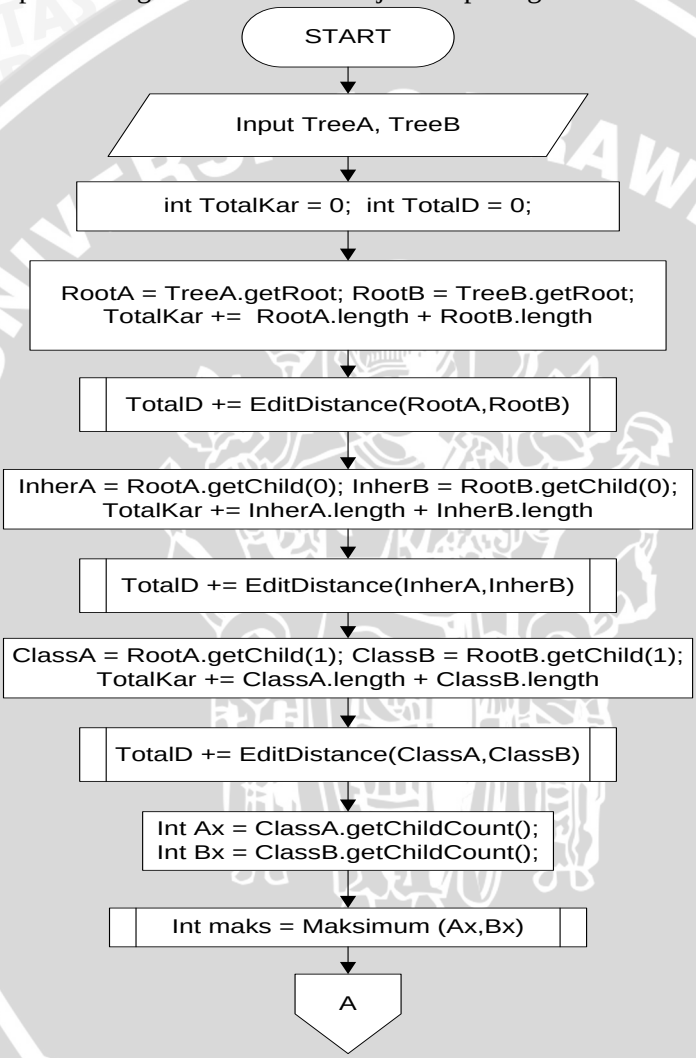
Gambar 3.9 Flowchart Ekstrak *Method*

Setelah array yang berbentuk FAMIX sudah dibuat kemudian dilakukan proses pembentukan *tree* dengan bantuan java GUI yang sudah terkandung dalam JDK1.6 yaitu komponen j*tree*. Proses pembentukan *tree* dapat dilihat pada gambar 3.10.

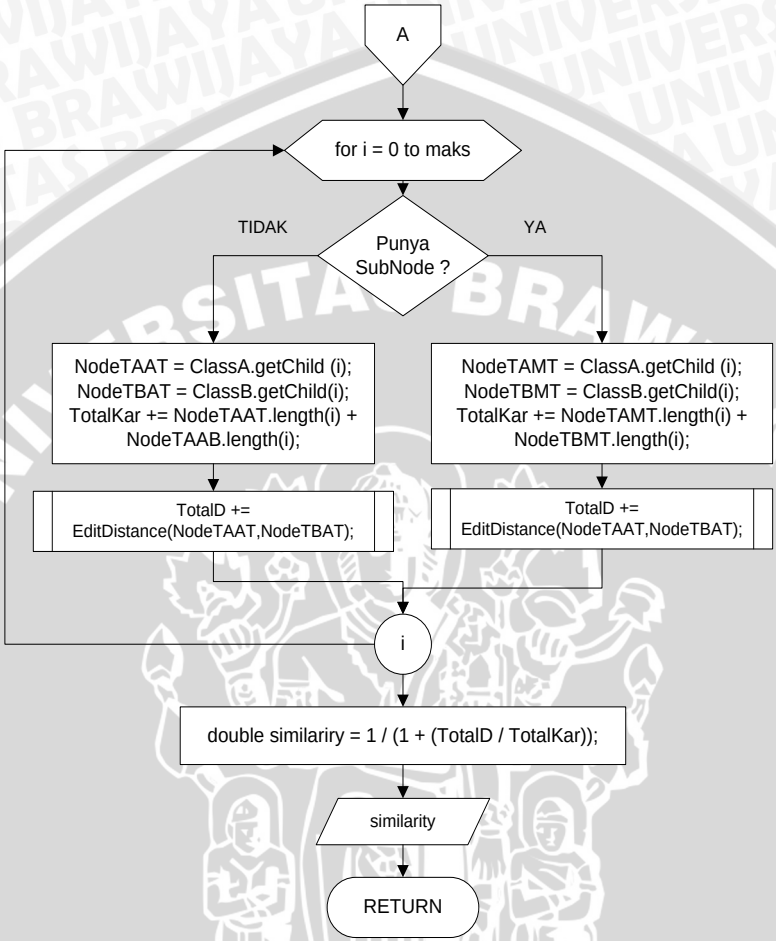


Gambar 3.10 Flowchart pembentukan *tree*

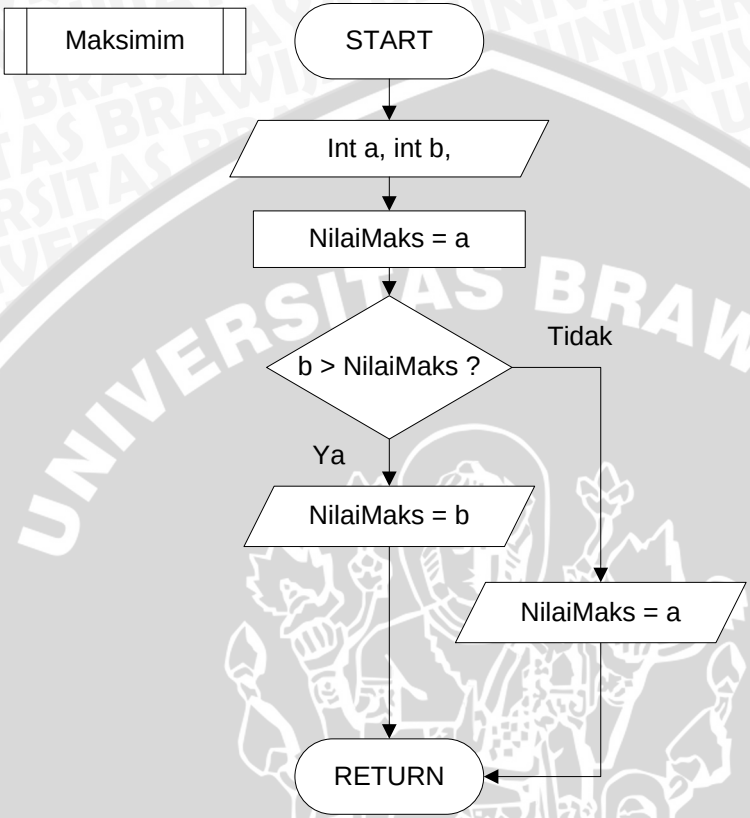
Dari *tree* yang terbentuk kemudian bandingkan isi *node* satu dengan *node* lain pada *tree* yang lain dengan menggunakan *editdistance*. Perbandingan pertama tanpa membandingkan isi *node*, dan perbandingan kedua dengan membandingkan isi *node* antar *tree*. Flowchart perbandingan antar *node* ditunjukkan pada gambar 3.11.



Gambar 3.11 Flowchart *EditDistance* tiap *node*.

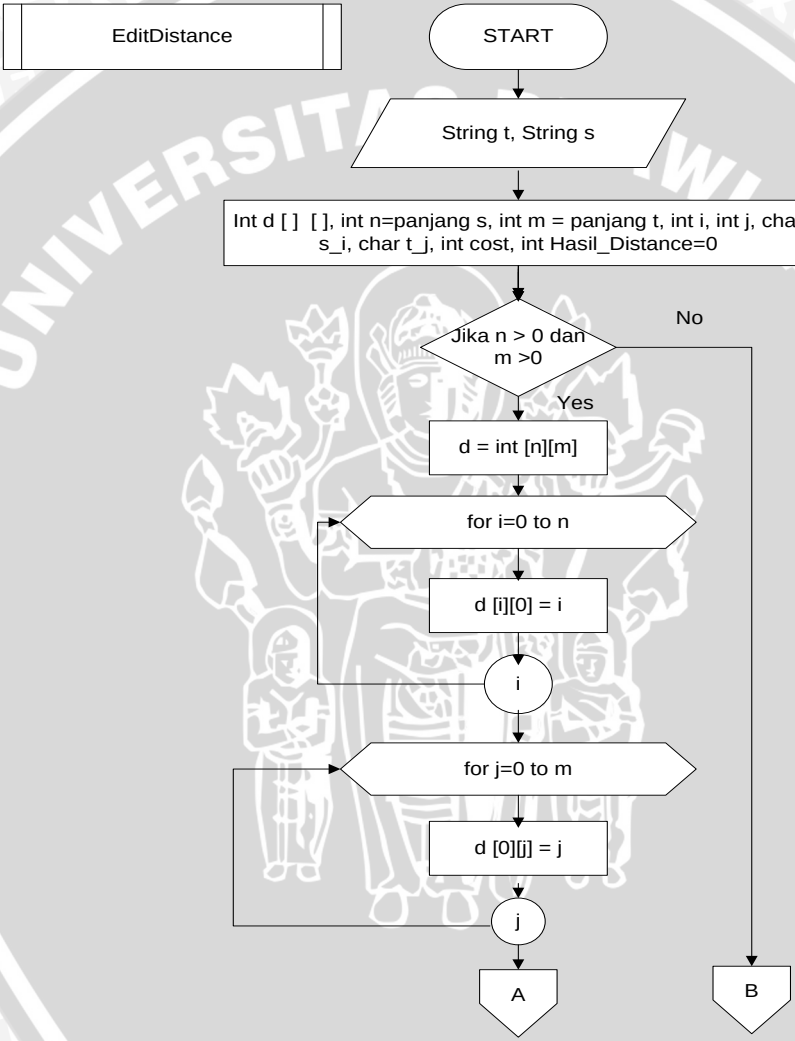


Gambar 3.12 Flowchart *EditDistance* tiap *node2*.

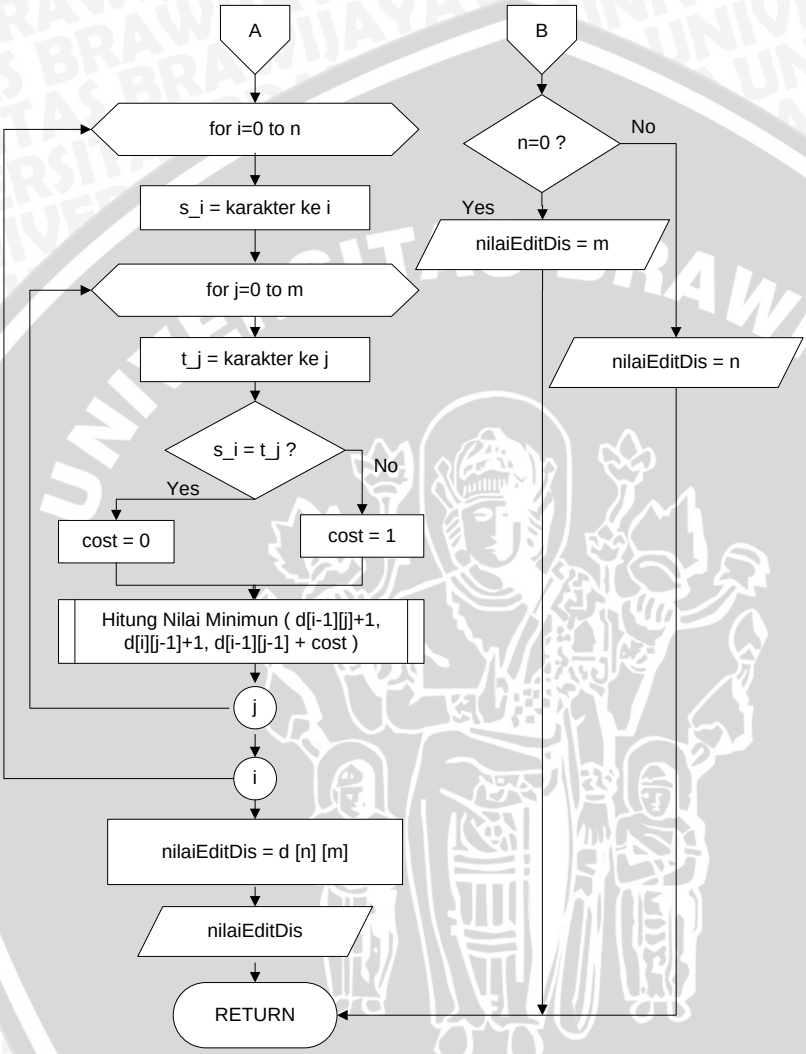


Gambar 3.13 Flowchart mencari nilai maksismum

Dalam proses perbandingan tiap *node*, terdapat metode untuk menghitung nilai *editdistance* untuk tiap *node* yang dibandingkan. Flowchart metode *editdistance* dapat dilihat pada gambar 3.14 dan 3.15.

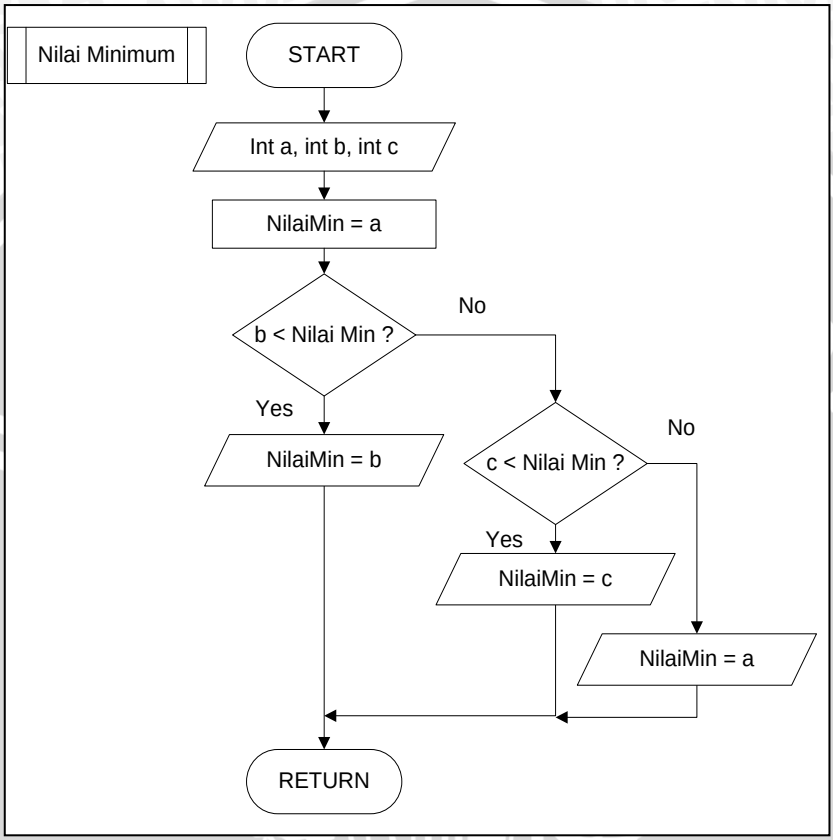


Gambar 3.14 Flowchart Hitung nilai *EditDistance* untuk *String*.



Gambar 3.15 Flowchart Hitung nilai *EditDistance* untuk *String* II

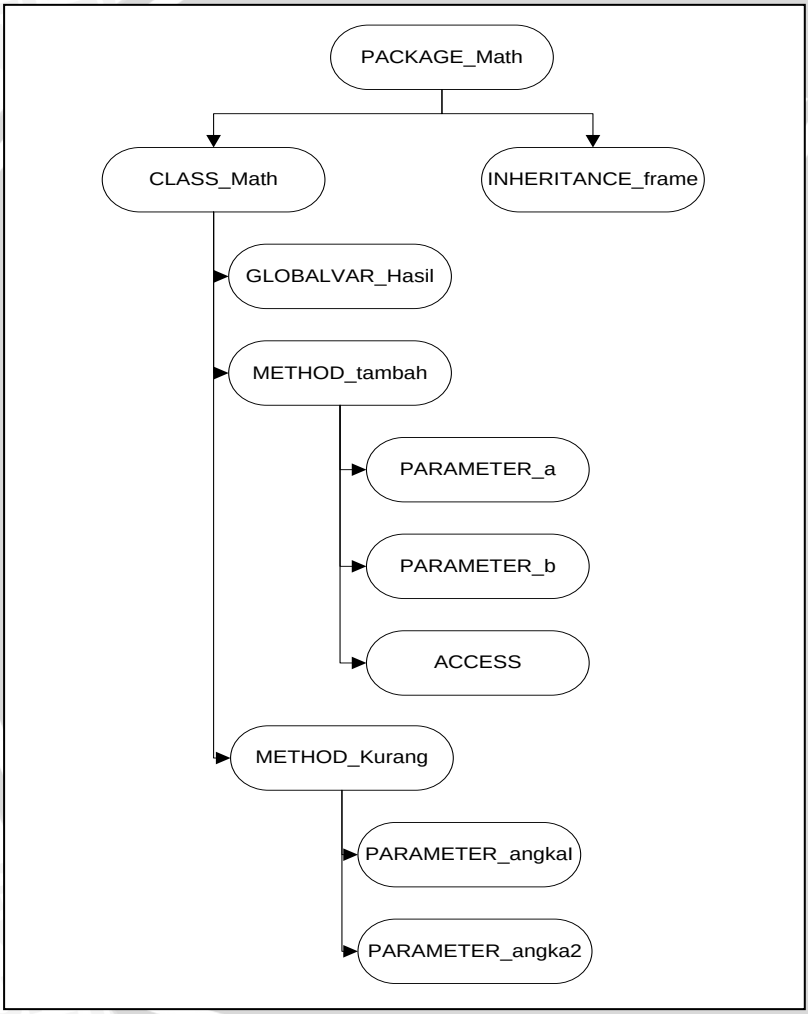
Sedangkan prosedur untuk menghitung nilai minimum dapat dilihat pada gambar 3.16.



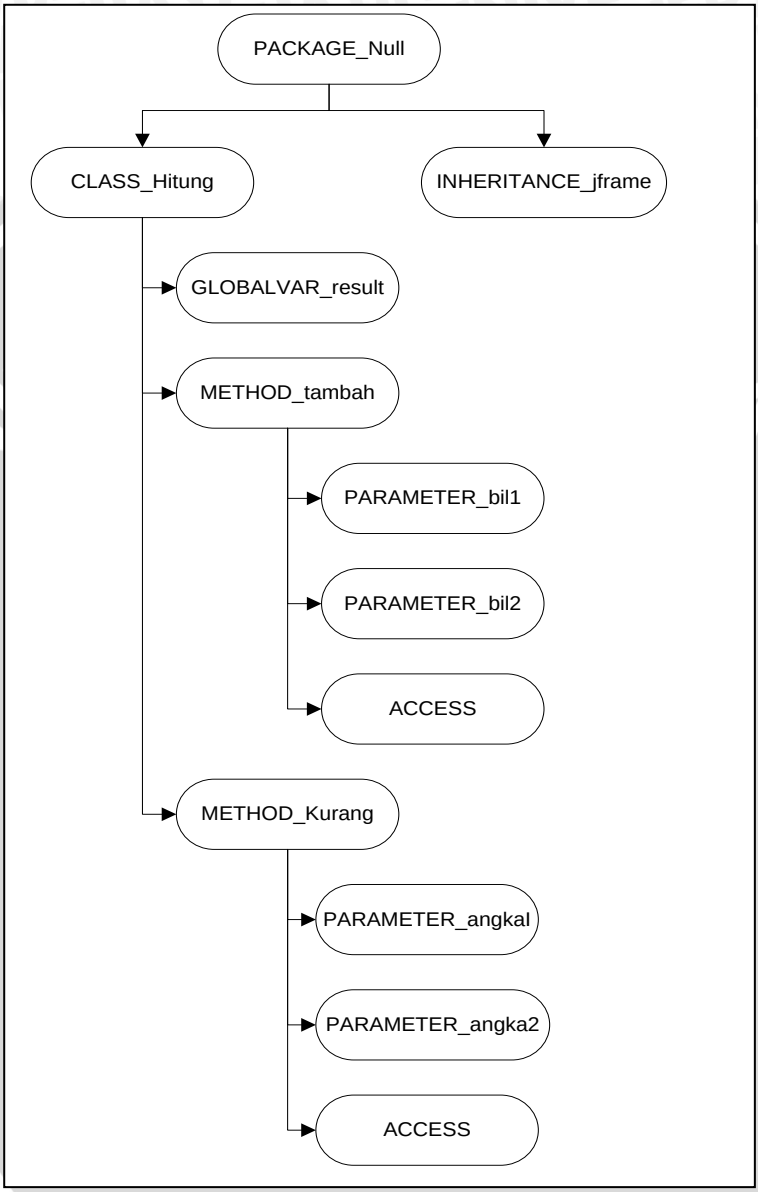
Gambar 3.16 Flowchart cari nilai minimum

3.3 Contoh Perhitungan

Pada subbab ini akan dijelaskan contoh perhitungan secara manual dari *metode tree editdistance* dalam mendeteksi perbedaan diatara *tree*. Sebagai ilustrasi maka akan dicari nilai *editdistance* antara *tree* A yang digambarkan pada gambar 3.17 dan *tree* B yang digambarkan pada gambar 3.18.



Gambar 3.17 Ilustrasi *Tree* A



Gambar 3.18 Ilustrasi *Tree* B

Proses perhitungan :

1. Perhitungan *EditDistance* antar root (package)

   S1 = PACKAGE_Math (*Tree* A) ∑S1 = 12
   S2 = PACKAGE_Null (*Tree* B) ∑S2 = 12
   d(S1,S2) = 4

2. Perhitungan *node* inheritance
   S1 = INHERITANCE_frame (*Tree* A) ∑S1 = 17
   S2 = INHERITANCE_ jframe (*Tree* B) ∑S2 = 18
   d(S1,S2) = 1

3. Perhitungan *node* class
   S1 = CLASS_Math (*Tree* A) ∑S1 = 10
   S2 = CLASS_Hitung (*Tree* B) ∑S2 = 12
   d(S1,S2) = 5

4. Perhitungan *node* global variable
   S1 = ATTRIBUTE_Hasil (*Tree* A) ∑S1 = 15
   S2 = ATTRIBUTE_result (*Tree* B) ∑S2 = 16
   d(S1,S2) = 4

5. Perhitungan Method dan *node* dalam tiap Method
   ➔ Method Tambah

   S1 = METHOD_tambah (*Tree* A) ∑S1 = 13
   S2 = METHOD_tambah (*Tree* B) ∑S2 = 13
   d(S1,S2) = 0

   S1 = PARAMETER_a (*Tree* A) ∑S1 = 11
   S2 = PARAMETER _bil1 (*Tree* B) ∑S2 = 14
   d(S1,S2) = 4

S1 = PARAMETER _b (*Tree* A) ∑S1 = 11
S2 = PARAMETER _bil2 (*Tree* B) ∑S2 = 14
d(S1,S2) = 3

S1 = ACCESS (*Tree* A) ∑S1 = 6
S2 = ACCESS (*Tree* B) ∑S2 = 6
d(S1,S2) = 0

➔ Method Kurang

Karena pada *tree* A tidak memiliki cabang untuk method kurang, maka hasil perhitungan *editdistance* adalah jumlah *string* pada *node* method kurang pada *tree* B.

S1 = METHOD_Kurang (*Tree* A) ∑S1 = 13
S2 = METHOD_Kurang (*Tree* B) ∑S2 = 13
d(S1,S2) = 0

S1 = PARAMETER_angka1 (*Tree* A) ∑S1 = 16
S2 = PARAMETER_angka1 (*Tree* B) ∑S2 = 16
d(S1,S2) = 0

S1 = PARAMETER_angka1 (*Tree* A) ∑S1 = 16
S2 = PARAMETER_angka1 (*Tree* B) ∑S2 = 16
d(S1,S2) = 0

S1 = ACCESS (*Tree* A) ∑S1 = 6
S2 = (*Tree* B) ∑S2 = 0
d(S1,S2) = 6

Jumlah String T1 = 12+17+10+15+13+11+11+6+13+16+16+6 = 146
Jumlah String T2 = 12+18+12+16+13+14+14+6+13+16+16+0 = 150
Maksimum = Jumlah String T2 = 150.

Setelah dihitung nilai *editdistance* untuk tiap *node*, kemudian dihitung nilai seluruh *editdistance* tiap *node*, dengan membandingkan nilai seluruh *editdistance* dengan jumlah *string* pada *tree* A dan *tree* B.

$$EditDistance\ (T1,T2) = \frac{\sum d(T1,T2)}{\sum(|V1| + |V2|)}$$

$$= \frac{4+1+5+4+0+4+3+0+0+0+0+6}{150}$$

$$= 27/150 = \mathbf{0{,}18} \qquad \textbf{(2.3)}$$

$$SimEditDistance(T1,T2) = \frac{1}{1 + StringEditDistance(T1,T2)}$$

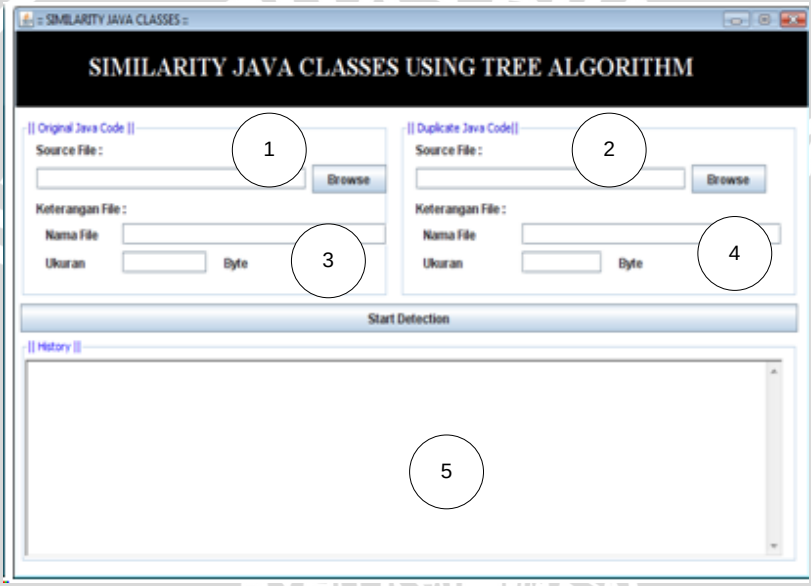
$$= 1 / (1 + 0{,}18)$$

$$= \mathbf{0{,}8475} \qquad \textbf{(2.4)}$$

Sehingga besar nya nilai similaritas antara *Tree* A dan *Tree* B adalah 0,8475 x 100 % = **84,75** %.

3.4 Perancangan *User Interface*

Rancangan tampilan *user interface* untuk aplikasi pendeteksi kesamaan kode program bahasa java terdiri dari satu halaman utama yang terdiri dari 3 tombol utama yaitu 2 tombol untuk load java file dan 1 tombol untuk memulai deteksi. Form utama juga dilengkapi dengan history yang berisikan langkah-langkah dan hasil dari proses pendetesi kesamaan kode porgram. Tampilan dari perancangan *user interface* dapat dilihat pada gambar 3.19.



Gambar 3.19 Rancangan *User Interface*

Keterangan gambar :

1. Tombol untuk load original java file.
2. Tombol untuk load duplicate java file.
3. Informasi mengenai original java file yang telah di load.
4. Informasi mengenai duplicate java file yang terlah di load.
5. History, unutk menampilkan proses pendeteksian kesamaan diantara java file.

3.5 Pengujian *Source Code*

Dalam penelitian ini, akan dilakukan uji coba terhadap 10 buah *source code* dengan cara membandingkan dengan 6 buah *source code* yang dianggap sebagai plagiat. *Source code* yang dianggap palgiat meliputi :

1. *Source code* sama, tidak mengalami perubahan apapun. Percobaan ini diinisialisasi sebagai percobaan A.
2. *Source code* yang telah dimodifikasi dengan menambahkan *constructor* dan *attibute* pada *source code* asli. Percobaan ini diinisialisasi sebagai percobaan B.
3. *Source code* yang telah dimodifikasi dengan menambahkan *method* atau *function* kedalam *source code* asli dan di panggil pada *main class*, pada *source code* asli. Percobaan ini diinisialisasi sebagai percobaan C.
4. *Source code* yang sama namun nama *class, package,variable* dan *method* diubah. Percobaan ini diinisialisasi sebagai percobaan D.
5. *Source code* yang sama sekali berbeda dengan *source code* yang diuji. Percobaan ini diinisialisasi sebagai percobaan E.
6. *Source Code* sama, namun *source code* I ditambahkan method yang mengandung perulangan *while* dan *source code* II ditambahkan method yang mengandung perulangan *for*. Percobaan ini diinisialisasi sebagai percobaan F.

Proses pengambilan data dari proses pengujian *source code* dapat dilihat pada table 3.1.

Tabel 3.1 Tabel Pengujian *Source code*

Source Code	Tingkat Kemiripan (%)					
	A	B	C	D	E	F

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implmentasi

Lingkungan yang digunakan untuk mengimplementasikan sistem pendeteksi plagirisme *source code* java meliputi lingkungan perangkat keras (*hardware*) dan lingkungan perangkat lunak (*software*).

4.1.1 Lingkungan Perangkat Keras (*Hardware*)

Perangkat keras yang digunakan untuk membuat sistem pendeteksi plagiarisme *source code* pada java adalah *laptop* dengan spesifikasi :

1. Prosessor Intel Pentium(R) Dual-Core CPU T4300 @2.10 GHz 2.10 GHz.
2. Memory RAM 2 GB DDR 2.
3. Kapasitas Hardisk 320 GB.

4.1.2 Lingkungan Perangkat Lunak (*Software*)

Perangkat lunak yang digunakan untuk membuat sistem pendeteksi plagiarisme *source code* pada java adalah :

1. Sistem Operasi Windows Vista ™ Ultimate 32-Bit
2. NetBeans IDE 6.8
3. Java SDK 1.6.0_19

4.2 Implementasi Program

Implementasi program dibuat berdasarkan perancangan proses yang dijelaskan pada subbab 3.2. Pada subbab 4.2 akan dijelaskan implementasi program dari proses – proses tersebut

4.2.1 Struktur Struktur Data

Pada pembuatan sistem ini akan dibentuk beberapa kelas-kelas utama yang menerapkan tiap proses dari sistem. Kelas-kelas tersebut antara lain: FMain, JavaParserMethod, ConvertToFamix, *Tree*, *Tree*Comparation,dan EditDistance.

1. Kelas FMain

Kelas FMain merupakan kelas yang menjadi interface dalam sistem pendeteksi plagiarisme *source code* program java. Kelas FMain merupakan inti dari semua kelas yang ada, karena setipa kelas lain akan dipanggil dan dieksekusi pada kelas ini. Pada kelas ini terdapat juga fungsi yang mem-*filter file* yang akan dideteksi, hanya *file* yang berekstensikan *.java* yang dapat diinputkan.

2. Kelas JavaParserMethod

Kelas JavaParserMethod merupakan kelas yang berfungsi untuk memparsing *source code* java ke dalam bentuk AST*Node* (*AbstractSyntaxTree Node*). Kelas ini dibentuk dengan memanfaatkan *library* javaparser-1.0.8, sehingga proses parsing *source code* menjadi lebih cepat. Pada kelas ini terdapat beberapa metode seperti : getClassName(), getExtendsOrImplementsName(), get*Package*(), getGlobalvariable(),getConstructor(),dan getMethod().

Penggunaan *library* javaparser-1.0.8 ini memiliki persyaratan khusus, yaitu souce code yang akan diparsing harus benar atau tidak *error*, jika terdapat kesalahan maka proses parsing tidak akan bisa dilakukan.

3. Kelas ConvertToFamix

Kelas ConvertToFamix merupakan kelas yang berfungsi untuk merubah hasil parsing yang dilakukan pada kelas JavaParserMethod. Hasil parsing akan ditambahkan elemen FAMIX pada header. Pada kelas ini terdapat beberapa fungsi seperti: ConvertToFamix*Package*Name(),ConvertToFamixClassName(), ConvertToFamixInhertiDefinition(),ConvertToFamixGlobalVariable (),ConvertToFamixMethod(),ConvertToparameter(),dan ConvertToConstructor().

4. Kelas *Tree*

Kelas *Tree* merupakan kelas yang berfungsi untuk membentuk elemen *source code* yang telah dirubah ke dalam bentuk FAMIX ke dalam representasi *tree*. Setiap Elemen yang telah diparsing akan dibentuk sesuai dengan pembentukan aturan FAMIX.

Proses pembentukan *tree* dilakukan dengan memanfaatkan fungsi *JTree*() yang terdapat dalam SDK 1.6.0. metode yang terdapat dalam kelas *tree* antara lain: initPrimer(),initConstructor(), initGlobalVar(),initClass(),initMethod(),dan build(). *Tree* yang terbentuk akan ditampilkan kedalam Frame, sehingga memudahkan user dalam mengamati bentuk *tree* dari sebuah *source code*.

5. Kelas *Tree*Comparation

Kelas *Tree*Comparation merupakan kelas yang berfungsi untuk membandingkan tiap-tiap *node* yang terdapat pada masing-masing *tree*. Kelas *TreeComparation* digunakan untuk menmbandingakan struktur dari kedua *tree* yang dibandingkan. Setiap *node* pada *tree* akan dibandingkan jika memiliki sturktur yang sama, apabila suatu *node* tidak memiliki kesamaan struktur maka akan dibandingkan dengan string kosong.

Kelas *TreeComparation* juga mengitung besar perbedaan string tiap-tiap *node* pada tiap-tiap *tree*, serta menghitung jumlah karakter yang ada didalam *node* pada setiap *tree*. Proses menghitung perbedaan string pada *node* dilakukan dengan memanggil kelas EditDistance. Pada kelas ini terdapat fungsi yang menghitung besar nya persentase kesamaan diantara *source code* yang dibancingkan.

6. Kelas EditDistance

Kelas EditDistance merupakan kelas yang digunakan untuk menghitung jarak atau distance diantara string. Kelas EditDistance memiliki dua fungsi yaitu getEditDistance() dan Minimum(). Kelas EditDistance akan di panngil pada kelas *TreeComparation* untuk menghitung total perbedaan string pada *node*.

4.2.2 Tahap Parsing *Source code*

Tahap parsing *source code* terdapat pada kelas JavaParserMethod. Tahap parsing souce code dilakukan terhadap *file* java. Tahapan pertama dari proses ini adalah meng-*compile source code* inputan, fungsi ini terdapat pada *source code* 4.1. Tahapan selanjutnya adalah memanggil fungsi getClassName() yang terdapat pada *source code* 4.2, getExtendsOrImplementsName() yang terdapat pada *source code* 4.3, get*Package*() yang terdapat pada *source code* 4.4, getGlobalvariable() yang terdapat pada *source code*

4.5 ,getConstructor() yang terdapat pada *source code* 4.6, dan getFixMethod() yang terdapat pada *source code* 4.7, untuk mendapatkan elemen-elemen *source code*.

```
public void CompilationUnitProcedure
(FileInputStream fis){
        try {
            Cu = JavaParser.parse(fis);
        } catch (ParseException ex) {
            ex.printStackTrace();
        }
}
```

Source code 4.1 Proses Compile *Source code*

```
public String getClassName(){
    String ret ="";
    new ClassNameVisitor().visit(Cu, null);
    ret = ClassName;
    return ret;
}
```

Source code 4.2 Fungsi getClassName()

```
public String getExtendsOrImplementClass(){
String ret ="";
new ExtendsImplementsVisitor().visit(Cu, null);
ret = ExtendsOrImplementName;
return ret;
}
```

Source code 4.3 Fungsi getExtendsOrImplementsName()

```
public String getPackages(){
String ret ="";
try {
ret = Cu.getPackage().toString().replace("package
", "").replace(";", "");
}catch (NullPointerException ex){
     ret = "null";
}
return ret ;
}
```

Source code 4.4 Fungsi get*Package* ()

```
public String[] getGlobalVariable(){
int a = 0,b = 0;
for (int i = 0; i < ArrayOfVariable.length; i++) {
    if (!ArrayOfVariable[i].equals("")){
          a +=1;
    }
}
String [] glob= new String [a];
for (int i = 0; i < ArrayOfVariable.length; i++) {
     if (!ArrayOfVariable[i].equals("")){
         glob [b] = ArrayOfVariable[i];
         b+=1;
        }
    }
    return  glob;
}
```

Source code 4.5 Fungsi getGlobalvariable ()

```
public String GetConstructor(){
    String r="";
    new ConstructorVisitor().visit(Cu,null);
    r = ConstructorName;
    return r;
}
```

Source code 4.6 Fungsi getConstructor ()

```
public String [] getFixMethod(){
   for (int i = 0; i < FixMethodName.length; i++) {
        FixMethodName[i].replace("null", "");
   }
   return  FixMethodName;
}
```

Source code 4.7 Fungsi getFixMethod ()

4.2.3 Tahap Konversi ke Bentuk FAMIX

Tahap konversi ke bentuk FAMIX merupakan tahapan dimana elemen *source code* yang telah diparsing akan ditambahkan header berupa string FAMIX, sehingga menjadi bentuk FAMIX. Tahapan koneversi ini terdapat dalam Kelas ConvertToFamix. Metode yang terdapat pada tahapan ini antara lain: ConvertToFamix*Package*Name() dapat dilihat pada *source code* 4.8, ConvertToFamixClassName() dapat dilihat pada *source code* 4.9, ConvertToFamixInhertiDefinition() dapat dilihat pada *source code* 4.10, ConvertToFamixGlobalVariable() dapat dilihat pada *source code* 4.11, ConvertToFamixMethod() dapat dilihat pada *source code* 4.12, ConvertToparameter() dapat dilihat pada *source code* 4.13,dan ConvertToConstructor() dapat dilihat pada *source code* 4.14.

```
public String ConvertToFamix Package Name(String
Package Name){
        String ret ="";
        Famix Package = "PACKAGE_"+ Package Name;
        ret = Famix Package;
        return ret;
}
```

Source code 4.8 ConvertToFamix*Package*Name()

```
public String ConvertToFamixClassName (String
ClassName){
        String ret ="";
        FamixClass = "CLASS_"+ ClassName;
        ret = FamixClass;
        return ret;
    }
```

Source code 4.9 ConvertToFamixClassName()

```
public String ConvertToFamixInhertiDefinition
(String inher){
        String ret="";
        FamixInheritance = "INHERITANCE_"+ inher;
        ret = FamixInheritance;
        return ret;
 }
```

Source code 4.10 ConvertToFamixInhertiDefinition()

```
Public String []
ConvertToFamixGlobalVariable(String  global []){
String [] ret=null;
ret = new String[global.length];
for(int j=0;j<global.length;j++){
  ret [j] = "ATTRIBUTE_" + global[j].toString();
}
     return  ret;
}
```

Source code 4.11 ConvertToFamixGlobalVariable()

```
public String [] ConvertToFamixMethod(String  met
[]){
String [] ret=null;
ret = new String[met.length];
for(int j=0;j<met.length;j++){
  met[j] = met[j].substring(4, met[j].length());
  ret [j] = "METHOD_" + met[j].toString();
}
   return  ret;
}
```

Source code 4.12 ConvertToFamixMethod()

```
public String ConvertToparameter(String par){
        String ret="";
        FamixParameter = "PARAMETER_"+par;
        return ret;
    }
```

Source code 4.13 ConvertToparameter()

```
public String ConvertToConstructor (String c){
        if (c.equals("")){
        return "CONSTRUCTOR_null";
        }
        else{
        return "CONSTRUCTOR_"+c;
        }
    }
```

Source code 4.14 ConvertToConstructor()

4.2.4 Tahap Pembentukan *Tree*

Tahap pembentukan *tree* merupakan tahapan dinama elemen *source code* yang telah diparsing dan dibentuk ke FAMIX di representasikan sebagai *tree*.

Tahap awal dari pembentukan *tree* adalah proses mendeklarasikan *tree*, *node* untuk *root*, *node* untuk *package node* untuk inheritance, *node* untuk class dan *node* untuk *constructor*. *Source code* dari pembentukan variabel *node* tersebut dapat dilihat pada *source code* 4.15

```
public class Tree extends JFrame{
DefaultMutableTreeNode      root,PNode,CNode,INode,
ConsNode;
JTree tree;
JScrollBar jsbver;
JScrollBar jsbhor;
public Tree(){
super("TREE GENERATION");
setSize(600, 600);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
```

Source code 4.15 Deklarasi variabel *Node* inti

Tahap selanjutnya adalah mengisi *node* yang telah dideklarasikan dan menambahkan hubungan parent dan child pada tiap *node*. Proses tersebut dapat dilihat pada *source code* 4.16 sampai *source code* 4.20.

```
public void initPrimer(Object p, Object ih){
 root  = new DefaultMutableTreeNode("FAMIX MODEL");
 PNode = new DefaultMutableTreeNode(p);
 INode = new DefaultMutableTreeNode(ih);
 root.add(INode);
 root.add(PNode);
}
```

Source code 4.16 Deklarasi *root*, *node* inheritance dan *node package*

```
public void initConstructor(String cons){
    CNode.add(new DefaultMutableTreeNode(cons));
  }
```

Source code 4.17 Deklarasi *node* constructor.

```
public void initClass(String c){
      CNode = new DefaultMutableTreeNode(c);
      PNode.add(CNode);
  }
```

Source code 4.18 Deklarasi *node* class.

```
public void initGlobalVar(String [] gbl){
 for (int i = 0; i < gbl.length; i++) {
 if (!gbl[i].equals("null")){
 CNode.add(new DefaultMutableTreeNode(gbl[i]));
 }
}
}
```

Source code 4.19 Deklarasi *node* global variable.

```
public void initMethod (String [] met){
      try{
      for (int i = 0; i < met.length; i++) {
          //System.out.println(met[i]);
          if (!met[i].equals("null")){
          String [] mm = met[i].split("@");
   CNode.add(new DefaultMutableTreeNode(mm[0]));
   mm[1] = mm[1].replace("[", "").replace("]", "");

          // Parameter
          String NN [] = mm[1].split(",");
          for (int j = 0; j < NN.length; j++) {
              //System.out.println(NN[j]);
        if (j==0){
          CNode.getLastLeaf().add(new
DefaultMutableTreeNode("PARAMETER_"+NN[0]));
              }
              else{
                  DefaultMutableTreeNode bb = new
DefaultMutableTreeNode();
                  bb = (DefaultMutableTreeNode)
CNode.getLastLeaf().getParent();
```

```
                    bb.add(new
DefaultMutable TreeNode("PARAMETER_"+NN[j]));
                }
            }

            // INVOCATION
           try {
                DefaultMutable TreeNode dd = new
DefaultMutable TreeNode();
                dd = (DefaultMutable TreeNode)
C Node.getLastLeaf().getParent();
                dd.add(new
DefaultMutable TreeNode(mm[2]));
           } catch (Exception e) {
           }
            // ACCESS
         try {
                DefaultMutable TreeNode dd2 = new
DefaultMutable TreeNode();
                dd2 = (DefaultMutable TreeNode)
C Node.getLastLeaf().getParent();
                dd2.add(new
DefaultMutable TreeNode(mm[3]));
            } catch (Exception e) {
            }
            }
       }
       }catch(Exception ex){

       }
   }
```

Source code 4.20 Deklarasi *node* metode dan *subtree*.

Setelah semua *node* telah didelarasikan, langkah selanjutnya adalah memasukkan semua *node* ke dalam *tree* yang telah dideklarasikan pada *source code* 4.15. Source deklarasi *tree* dapat dilihat pada *source code* 4.21.

```
public void bulid(){
 tree = new JTree(root);
 getContentPane().add(tree, BorderLayout.CENTER);
 JScrollPane scrollpane = new JScrollPane(tree);
 getContentPane().add(scrollpane, "Center");
}
```

Source code 4.21 Deklarasi *tree* dari komponen pembentuk

4.2.5 Tahap Perbandingan *Tree*

Tahap perbandingan *tree* merupakan tahap dimana struktur setiap *node* pada *tree* dibandingan dengan struktur tiap *node* pada *tree* yang lain. Langkah pertama yang dilakukan pada tahap ini adalah membandingkan *root,* dan child dari *root* yang meliputi inheritance dan *package*. Setiap *node* yang dibandiangkan akan diambil isi *node* kemudian dibandingakan dengan memanfaatkan kleas EditDistance. Proses perbandingan *root* dan child nya dapat dilihat pada fungsi sumEditDistance yang dapat dilihat pada *source code* 4.22.

```
public  int SumEditDistance(){
   int ret = 0;
   for (int i = 0; i < TreeA.getRowCount(); i++) {
   TreeA.expandRow(i);
   System.out.println("ROW ( "+ i +"
)"+TreeA.getPathForRow(i).toString().replace(" ",
""));
   }
   for (int j = 0; j < TreeB.getRowCount(); j++) {
   TreeB.expandRow(j);
   System.out.println("ROW ( "+ j +"
)"+TreeB.getPathForRow(j).toString().replace(" ",
""));
   }
   ////////// EditDistance untuk ROOT
   ret =
ed.getEditDistance(TreeA.getModel().getRoot().toStr
```

```
ing(),TreeB.getModel().getRoot().toString());
TotKarA +=
TreeA.getModel().getRoot().toString().length();
        TotKarB +=
TreeB.getModel().getRoot().toString().length();
        //System.out.println("d (ROOT) :
"+ed.getEditDistance(TreeA.getModel().getRoot().toS
tring(),TreeA.getModel().getRoot().toString()));

////////// Edit Distance untuk class (idx 1)
Inheritance (idx 0)
        ret +=
ed.getEditDistance(TreeA.getModel().getChild(TreeA.
getModel().getRoot(), 0).toString(),
TreeB.getModel().getChild(TreeB.getModel().getRoot(
), 0).toString());
        ret +=
ed.getEditDistance(TreeA.getModel().getChild(TreeA.
getModel().getRoot(), 1).toString(),
TreeB.getModel().getChild(TreeB.getModel().getRoot(
), 1).toString());
        TotKarA +=
TreeA.getModel().getChild(TreeA.getModel().getRoot(
), 0).toString().length();
        TotKarB +=
TreeB.getModel().getChild(TreeB.getModel().getRoot(
), 0).toString().length();
        TotKarA +=
TreeA.getModel().getChild(TreeA.getModel().getRoot(
), 1).toString().length();
        TotKarB +=
TreeB.getModel().getChild(TreeB.getModel().getRoot(
), 1).toString().length();
        //System.out.println(
ed.getEditDistance(TreeA.getModel().getChild(TreeA.
getModel().getRoot(), 0).toString(),
TreeB.getModel().getChild(TreeB.getModel().getRoot(
), 0).toString()) );
        //System.out.println(
ed.getEditDistance(TreeA.getModel().getChild(TreeA.
getModel().getRoot(), 1).toString(),
TreeB.getModel().getChild(TreeB.getModel().getRoot(
), 1).toString()) );
```

```
////////////// Edit Distance untuk Class
        DefaultMutableTreeNode CA =
(DefaultMutableTreeNode)
TreeA.getModel().getChild(TreeA.getModel().getRoot(
), 1);
        DefaultMutableTreeNode CB =
(DefaultMutableTreeNode)
TreeB.getModel().getChild(TreeB.getModel().getRoot(
), 1);
        // System.out.println(" CLASS " +
CA.getChildAt(0).toString());
        // System.out.println(" CLASS " +
CB.getChildAt(0).toString());
        ret +=
ed.getEditDistance(CA.getChildAt(0).toString(),
CB.getChildAt(0).toString());
        TotKarA +=
CA.getChildAt(0).toString().length();
        TotKarB +=
CB.getChildAt(0).toString().length();

        ////////// Edit Distance untuk Attribute /
Global Variabel
        int Ax = CA.getChildAt(0).getChildCount();
        int Bx = CB.getChildAt(0).getChildCount();

        ArrayList ALA = new ArrayList(); //
Menampung Global Var T1
        ArrayList ALB = new ArrayList(); //
Menampung Global Var T2

        ArrayList MLA = new ArrayList(); //
Menampung Method T1
        ArrayList MLB = new ArrayList(); //
Menampung Method T2

        // Jumlah GLobVar T1 dan T2 dan Jumlah
Method Pada T1 dan T2
        for (int i = 0; i <Ax; i++) {
            // Global variable
            if
```

```
(CA.getChildAt(0).getChildAt(i).isLeaf()){

ALA.add(CA.getChildAt(0).getChildAt(i).toString());
            }
            // Method
            else{
                int z =
CA.getChildAt(0).getChildAt(i).getChildCount();
                String x =
CA.getChildAt(0).getChildAt(i).toString();
                for (int j = 0; j < z; j++) {
                    x += "@"+
CA.getChildAt(0).getChildAt(i).getChildAt(j);
                }
                MLA.add(x);

            }
        }

        for (int j = 0; j < Bx; j++) {
            // Global variable
           if
(CB.getChildAt(0).getChildAt(j).isLeaf()){

ALB.add(CB.getChildAt(0).getChildAt(j).toString());
            }
           // Method
           else{
                int zz =
CB.getChildAt(0).getChildAt(j).getChildCount();
                String xx =
CB.getChildAt(0).getChildAt(j).toString();
                for (int n = 0; n < zz; n++) {
                   xx += "@" +
CB.getChildAt(0).getChildAt(j).getChildAt(n);
                }
                MLB.add(xx);
           }
        }
        Object ALAA [] = ALA.toArray();
        Object ALBB [] = ALB.toArray();
        Object MLAA [] = MLA.toArray();
        Object MLBB [] = MLB.toArray();
```

```
        for (int i = 0; i < MLAA.length; i++) {
             System.out.println("MLAA : " +
MLAA[i].toString());
        }
        for (int i = 0; i < MLBB.length; i++) {
            System.out.println("MLBB : " +
MLBB[i].toString());
        }
        ret += distanceAttribute(ALAA, ALBB);
        ret += distanceMethod(MLAA, MLBB);
        return ret;
    }
```

Source code 4.22 Fungsi sumEditDistance()

Selain menghitung *root* berserta child nya, didalam fungsi sumEditDistance () terdapat *statement* yang memanggil fungsi untuk menghitung nilai editdistance untuk membandingkan attribute dan method yang ada pada *tree*. Fungsi untuk membandingkan attribute dideklarasikan dengan nama distanceAttribute() yang dapat dilihat pada *source code* 4.23, sedangkan fungsi untuk membandingkan method dan sub*tree* nya dideklarasikan dengan nama distanceMethod() yang dapat dilihat pada *source code* 4.24.

```
public int distanceAttribute(Object [] A1, Object
[] A2){
  int d = 0;
  int maks = Maksimum(A1.length, A2.length);
  for (int i = 0; i < maks; i++) {
        try{
          d += ed.getEditDistance(A1[i].toString(),
A2[i].toString());
        TotKarA += A1[i].toString().length();
        TotKarB += A2[i].toString().length();
        }
        catch(Exception e){
        if ( A1.length == maks){
      d += ed.getEditDistance(A1[i].toString(),"");
      TotKarA += A1[i].toString().length();
        }
        else{
      d+= ed.getEditDistance("",A2[i].toString());
```

```
      TotKarB += A2[i].toString().length();
      }   }
        }
        return d;
    }
```

Source code 4.23 Fungsi distanceAttribute()

```
public int distanceMethod(Object[] M1, Object []
M2){
int dis = 0;
int maks = Maksimum(M1.length,M2.length);
String M1A []=null;
String M2B []=null;
for (int i = 0; i < maks; i++) {
try{
///////Edit Distance Nama Method
M1A  = M1[i].toString().split("@");
M2B  = M2[i].toString().split("@");
System.out.println("String M1A : " + M1A[0]);
System.out.println("String M2B : " + M2B[0]);
dis += ed.getEditDistance(M1A[0].toString(),
M2B[0].toString());
TotKarA += M1A[0].toString().length();
TotKarB += M2B[0].toString().length();

/// Sub Tree dr Method if Tree Memiliki bentuksama
int x = Maksimum(M1A.length, M2B.length);
for (int j = 1; j < x; j++) {
try{
System.out.println("M11A : " + M1A[j]);
System.out.println("M22B : " + M2B[j]);
dis +=
Selection(M1A[j].toString(),M2B[j].toString());
}
catch(Exception e){
if (M1A.length == x ){
System.out.println("M11A : " + M1A[j]);
System.out.println("M22B : " + "");
dis += Selection(M1A[j].toString(),"");
}
else{
System.out.println("M11A : " + "");
```

```
System.out.println("M22B : " + M2B[j]);
dis += Selection("",M2B[j].toString());
}
}
}
}catch(Exception e){
if (M1.length == maks){
M1A  = M1[i].toString().split("@");
System.out.println("String M1A : " + M1A[0]);
System.out.println("String M2B : " + "");
dis += ed.getEditDistance(M1A[0].toString(), "");
TotKarA += M1A[0].toString().length();
/// Sub Tree dr Method jk Tree Memiliki bentk sama
int x = Maksimum(M1A.length, M2B.length);
for (int j = 1; j < x; j++) {
try{
System.out.println("M11A : " + M1A[j]);
System.out.println("M22B : " + M2B[j]);
dis +=
Selection(M1A[j].toString(),M2B[j].toString());
}
catch(Exception ex){
if (M1A.length == x ){
System.out.println("M11A : " + M1A[j]);
System.out.println("M22B : " + "");
dis += Selection(M1A[j].toString(),"");
}
else{
System.out.println("M11A : " + "");
System.out.println("M22B : " + M2B[j]);
Selection("",M2B[j].toString());
}
}
}
}
else{
M2B  = M2[i].toString().split("@");
System.out.println("String M1A : " + "");
System.out.println("String M2B : " + M2B[0]);
dis += ed.getEditDistance("", M2B[0].toString());
TotKarB += M2B[0].toString().length();
/// Sub Tree dr Method jk Tree Memiliki bentuk sama
int x = Maksimum(M1A.length, M2B.length);
```

```
for (int j = 1; j < x; j++) {
try{
System.out.println("M11A : " + M1A[j]);
System.out.println("M22B : " + M2B[j]);
Selection(M1A[j].toString(),M2B[j].toString());
}
catch(Exception exc){
if (M1A.length == x ){
System.out.println("M11A : " + M1A[j]);
System.out.println("M22B : " + "");
 dis += Selection(M1A[j].toString(),"");
 }
 else{
 System.out.println("M11A : " + "");
 System.out.println("M22B : " + M2B[j]);
 Selection("",M2B[j].toString());
 } } } } } }
 return dis;
 }
```

Source code 4.24 Fungsi distanceMethod()

Proses perhitungan *Edit Distance* pada *node* method dan sub*tree* nya harus melalui fungsi selection(), hal itu bertujuan untuk memfilter *node* yang memiliki string null, karena string yang mengandung nilai null tidak akan dibandingkan dan diganti dengan string kosong. Fungsi selection dapat dilihat pada *source code* 4.25.

```
public int Selection (String A, String B){
int dis = 0;
//Periksa apakah ada node yang ber (PARAMETER_null)
// Jika ada maka node diganti dengan sting kosong
if
(!((A.equals("PARAMETER_null"))&&(B.equals("PARAMET
ER_null")))){
dis += ed.getEditDistance(A,B);
TotKarA += A.toString().length();
TotKarB += B.toString().length();
}
else if
(!(A.contains("PARAMETER_null"))&&(B.contains("PARA
METER_null"))){
dis += ed.getEditDistance(A.toString(), "");
TotKarA += A.toString().length();
```

```
}
else if ((A.contains("PARAMETER_null"))&&(!
B.contains("PARAMETER_null"))){
dis += ed.getEditDistance("", B.toString());
TotKarB += B.toString().length();
}
        return dis;
}
```

Source code 4.25 Fungsi Selection()

Fungsi Maksimum() pada tahap ini juga diperlukan untuk mencari nilai dari dua inputan. Fungsi maksimum digunanan dalam fungsi distanceAttribute() dan fungsi distanceMethod(). Fungsi maksimum dapat dilihat pada *source code* 4.26.

```
public int Maksimum(int a, int b){
        int maks = a;
        if (b>maks){
            maks = b;
        }
        return maks;
    }
```

Source code 4.26 Fungsi Maksimum()

4.2.6 Tahap Perhitungan *Edit Distance*

Tahap perhitungan *Edit Distance* merupakan tahap dimana string dalam setiap *node* pada *tree* yang akan dibandingkan dengan *node* yang memiliki kedudukan sama dalam *tree* yang dijadikan sebagai pembanding. Tahapan perhitungan *Edit Distance* berada pada kelas EditDistance, tahapan ini memiliki tiga buah fungsi utama yaitu fungsi Minimum(), fungsi getMak() dan fungsi getEditDistance(). Fungsi Minimum dapat dilihat pada *source code* 4.27, fungsi getMak() dapat dilihat pada *source code* 4.28, dan fungsi getEditDistance() dapat dilihat pada *source code* 4.29.

```
public int Minimum (int a, int b, int c) {
    int mi;
    mi = a;
    if (b < mi) {
      mi = b;
    }
    if (c < mi) {
      mi = c;
    }
    return mi;
  }
```

Source code 4.27 Fungsi Minimum()

```
public double getMak(String a, String b){
        double r =0;
        int p_a = a.length();
        int p_b = b.length();

        r = p_a;
        if (p_b>r){
            r = p_b;
        }
        return r;
  }
```

Source code 4.28 Fungsi getMak()

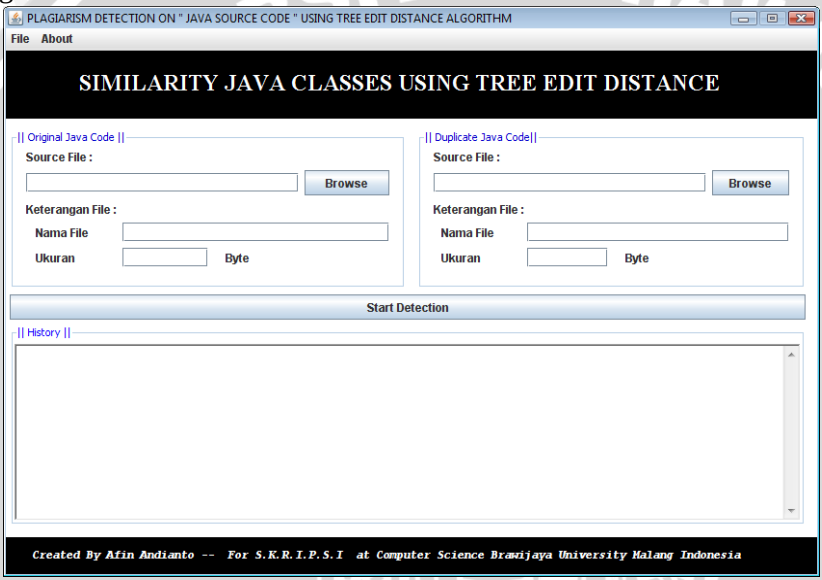
```
public int getEditDistance(String s, String t){
int d[][]; // inisialisasi matrix
int n; // panjang string s
int m; // panjang string t
int i; // indeks perulangan untuk string s
int j; // indeks perulangan untuk string t
char s_i; // karakter ke - i pada string s
char t_j; // karakter ke - j pada string t
int cost; // cost
System.out.println("String I \t: " + s +" PANJANG
SI : " + s.length());
System.out.println("String II\t: " + t +" PANJANG
S2 : " + t.length());
t.length());
n = s.length ();
m = t.length ();
```

```
if (n == 0) {
return m;
}
if (m == 0) {
return n;
}
// Inisialisasi matrik 2x2 dengan batas n dan m
d = new int[n+1][m+1];
// isi matrik 1x0 dengan nilai 0.. panjang n
for (i = 0; i <= n; i++) {
d[i][0] = i;
        }
// isi matrik 0x1 dengan nilai 0.. panjang m
for (j = 0; j <= m; j++) {
d[0][j] = j;         }
// Bandingkan tiap karakter pada matrix
for (i = 1; i <= n; i++) {
s_i = s.charAt (i - 1);
for (j = 1; j <= m; j++) {
t_j = t.charAt (j - 1);
// jika nilai sama hasilkan nilai 0
if (s_i == t_j) {
cost = 0;
}
// jika tidak sama hitung jarak
else {
cost = 1;
}
d[i][j] = Minimum (d[i-1][j]+1, d[i][j-1]+1, d[i-
1][j-1] + cost);
}     }
        return d[n][m];
}
```

Source code 4.29 Fungsi getEditDistance()

4.3 Implementasi Antarmuka (*Interface*)

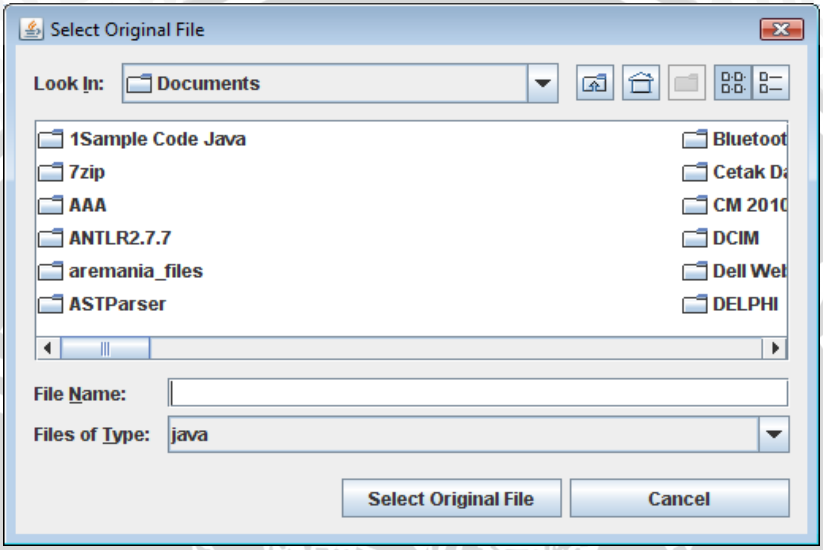
Implementasi interface dibuat berdasarkan rancangan pada subbab 3.4. Pada form utama terdapat dua buah panel masing-masing yang berisikan *field* "*source file"* dan *button "browse"* yang digunakan untuk mengisikan inputan *original file* dan *duplicate file*. Elemen lain yang ada didalam form main yaitu *button "Start Detection"* dan *textarea sebagai "hisory"*. Interface utama dari program pendeteksi plagiarisme *source code* java dapat dilihat pada gambar 4.1.



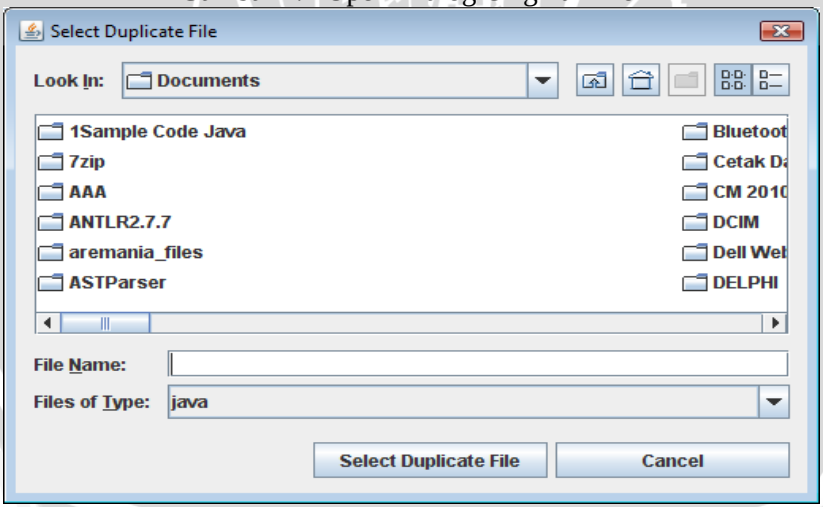
Gambar 4.1 Form Utama

Pada *panel* "*Original Java Code*" *field "source file"*, dapat diisi secara manual atau dengan menekan tombol *"browse"*. Field *"source file"* digunakan untuk menentukan *file* java yang dianggap sebagai *original file*. Jika tombol "*browse*" pada panel ini ditekan, maka akan muncul *dialog box "open"* yang ditampilkan pada gambar 4.2. *User* akan melakukan pemilihan dokumen yang akan dijadikan sebagai *file original*. File yang dipilih dari open dialog ini hanya file java yang berekstensikan .java. Jika proses pemilihan dokumen sudah selesai dilakukan, maka field akan berisikan *path* dari *file* java yang telah diinputkan. Selain *field* "*source file*", field

"*nama file*" dan "*ukuran file*" juga langsung keluar ketika proses pemilihan selesai. Panel "*Duplicate Java Code*" memiliki sifat yang sama dengan panel "*Original Java Code*". *Dialog box* untuk memilih *duplicate file* dapat dilihat pada gambar 4.3.

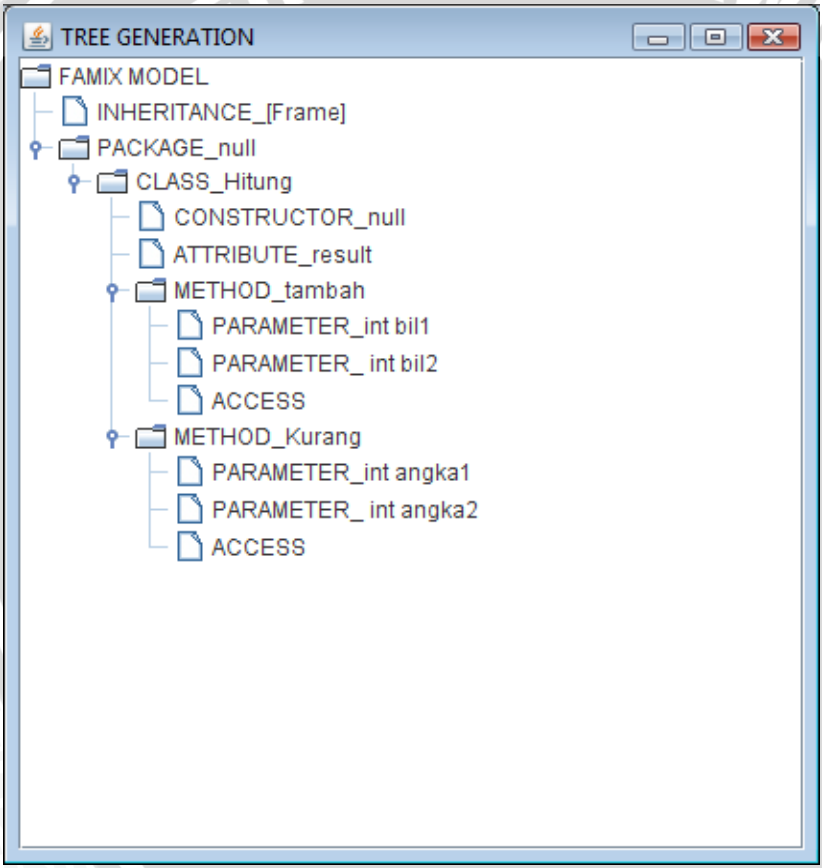


Gambar 4.2 Open Dialog Original File

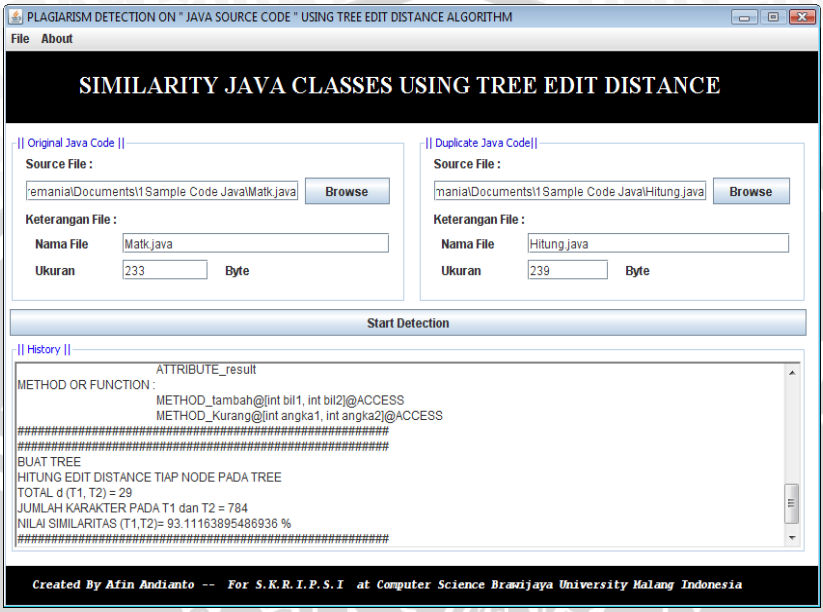


Gambar 4.3 Open Dialog Duplicate File

Proses pendeteksi kesamaan diantara *source code* java bisa dilakukan apabila user telah menginputkan dua buah file yang dianggap sebagai *original file* dan *duplicate file*. Tombol "Start Detection" digunakan untuk memulai proses pendeteksian diantara *source code* satu dengan *source code* yang lain. Proses pendeteksian dimulai dengan pembentukan *tree* yang akan muncul pada *frame* seperti gambar 4.4. Jika kedua *tree* sudah terbentuk maka proses selanjutnya adalah menampilkan langkah dan hasil pendeteksian kedalam "*history*". Gambar hasil pendeteksian dapat dilihat pada gambar 4.5.



Gambar 4.4 Pembentukan *Tree*



Gambar 4.5 Hasil Pendeteksian

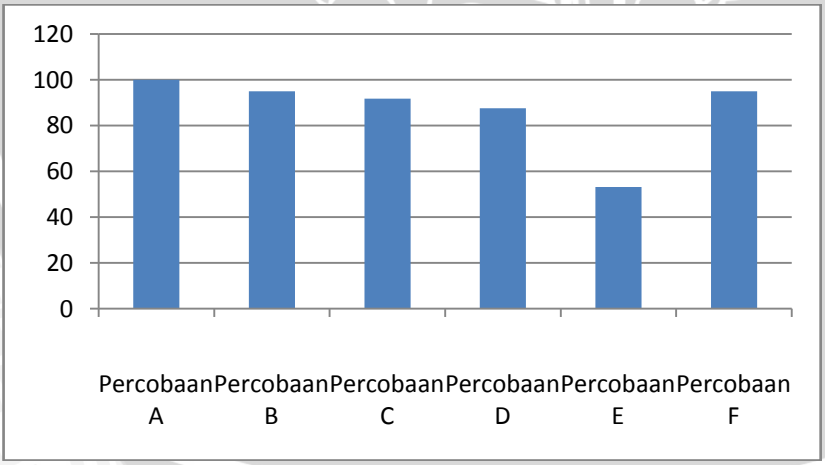
4.4 Analisa Hasil

4.4.1 Hasil Ujicoba dan Analisa Hasil

Hasil ujicoba aplikasi dalam mendeteksi plagiarisme *source code* pada bahasa java dapat dilihat pada tabel 4.1 dan grafik hasil uji coba dapat dilihat pada gambar 4.6.

Tabel 4.1 Hasil Ujicoba Sistem

Source Code	Tingkat Kemiripan (%)					
	A	B	C	D	E	F
1	100	91,37	88,57	84,17	53,42	94,13
2	100	94,18	92,7	87,43	53,72	94,66
3	100	97,14	91,26	89,62	53,42	93,61
4	100	97,56	94,81	89,53	53,7	96,57
5	100	95,96	90,31	90,1	53,08	98,62
6	100	95,26	91,2	84,66	53,17	91,12
7	100	95,68	89,71	89,01	52,45	98,52
8	100	89,08	90,96	88,19	54,2	98,88
9	100	95,8	94,24	86,47	52,09	90,31
10	100	97,48	93,73	85,89	52,02	92,87
AVERAGE	100	94,951	91,749	87,507	53,127	94,929



Gambar 4.6 Grafik Hasil Ujicoba

Keterangan Percobaan :

1. Percobaan A : kedua *source code* sama, tidak mengalami perubahan apapun.
2. percobaan B : *Source code* yang telah dimodifikasi dengan menambahkan *constructor* dan *attibute* pada *source code* asli.
3. Percobaan C : *Source code* yang telah dimodifikasi dengan menambahkan *method* atau *function* kedalam *source code* asli dan di panggil pada *main class*, pada *source code* asli.
4. Percobaan D : *Source code* yang sama namun nama *class*, *package*,*variable* dan *method* diubah.
5. Percobaan E : *Source code* yang sama sekali berbeda dengan *source code* yang diuji.
6. Percobaan F : *Source Code* sama, namun *source code* I ditambahkan method yang mengandung perulangan *while* dan *source code* II ditambahkan method yang mengandung perulangan *for*.

Berdasarkan hasil uji coba pada tabel 4.1 dan grafik pada gambar 4.6 diketahui bahwa hasil untuk ujicoba untuk percobaan A memiliki tingkat keberhasilan 100 persen, hal tersebut menunjukkan tidak terdapat kesalahan pada sistem. Hasil tersebut dipengaruhi dengan baiknya proses parsing yang dilakukan dengan bantuan java parser. Pada percobaan B nilai rata-rata yang dihasilkan mencapai 94,95 persen, hal ini menunjukkan bahwa penambahan *attribute* dan *constructor* pada source code tidak akan berdampak besar terhadap sistem dalam mendeteksi plagiat.

Percobaan C nilai rata-rata yang dihasilkan adalah 91,75 persen, percobaan ini menggambarkan bahwa penambahan satu *method* atau *function* tidak akan berpengaruh besar terhadap proses mendeteksi hal ini dikarenakan struktur *tree* yang terbentuk tidak akan berbeda banyak dengan struktur *tree source code* yang asli. Percobaan D menghasilkan nilai rata-rata 87,50 persen, hal ini membuktikan bahwa pergantian nama *variable* dan *method* menghasilkan nilai yang baik. Jika perbandingan *node* hanya dilakukan sampai struktur *tree* saja, maka akan menghasilkan nilai 100 persen, untuk itulah diperlukan proses perbandingan isi *node* yang terdapat pada struktur *tree*.

Metode *Tree Edit Distance* menghasilkan nilai yang cukup besar yaitu 53,127 % ketika membandingkan dua *source code* yang berbeda hal tersebut dapat dilihat pada percobaan E. Hasil ujicoba pada percobaan E dipengaruhi struktur dan isi *node* dari *tree* yang dibandingkan. *Tree* yang dibandingkan tidak sebatas strukturnya saja namun isi *node* pada *tree* juga dibandingkan.

Proses parsing akan sangat berpengaruh terhadap proses pendeteksian, karena proses parsing akan menentukan *node-node* yang ada pada *tree*. Proses parsing menghasilkan *attribute* yang berbeda ketika mendeteksi jenis perulangan '*while*' dan perulangan '*for*' sehingga modifikasi bentuk jenis perulangan dalam proses plagiat dapat terdeteksi dengan baik, hal itu dapat ditunjukkan pada percobaan F yang menghasilkan nilai rata-rata sebesar 94,93 persen.

Metode *Tree Edit Distance* menghasilkan nilai yang baik dengan rata-rata persentase 92,28 % ketika mendeteksi *source code* yang hanya memiliki sedikit perbedaan (percobaan B, percobaan C, percobaan D, percobaan F), hal ini membuktikan bahwa metode *Tree Edit Distance* tidak akan berpengaruh banyak ketika modifikasi hanya dilakukan pada *string* dalam *source code*. Hasil pendeteksian juga dipengaruhi perbedaan kompleksitas dari *source code* yang dibandingakan. Kompleksitas berpengaruh terhadap *tree* yang yang akan dibentuk. Hasil deteksi akan menghasilkan nilai yang kecil apabila perbedaan kompleksitas diantara *source code* besar, dan hasil pendeteksian menghasilkan nilai yang besar jika perbedaan kompleksitas kecil.

BAB V
PENUTUP

5.1 Kesimpulan

Kesimpulan yang dihasilkan pada penelitian ini adalah :

1. Telah dibuat sistem yang mengimplementasi metode *Tree Edit Distance* untuk sistem pendeteksi plagiarisme *source code* bahasa pemrograman java, dan telah dilakukan pengujian terhadap beberapa *source code*.
2. Algoritma *Tree Edit Distance* memiliki tingkat keberhasilan yang baik ketika mendeteksi plagiarisme diantara *source code.*
3. Nilai yang dihasilkan algoritma *Tree Edit Distance* dipengarui struktur dari *tree* yang dibandingkan. Semakin tinggi tingkat perbedaan diantara *tree* maka hasil pendeteksian juga semakin kecil.

5.2 Saran

Pengembangan lebih lanjut, disarankan proses parsing mencakup semua elemen *source code* termasuk comment, *if else statemnet* dan *loop statement*. Proses *string matching* diantara node perlu dicoba dengan metode lain sehingga tingakat akurasi menjadi lebih baik. Disamping itu, disarankan untuk mengembangkan sistem sehingga mampu mendeteksi kesamaan project yang terdiri dari beberapa kelas.

DAFTAR PUSTAKA

Aoe J. 1994. Computer algorithms string pattern matching strategies. John Wiley and Sons : New York.

Baker B S. 1995. *On finding duplication and near-duplication in large software systems*. In L. Wills,P. Newcomb, and E. Chikofsky, editors, *Second Working Conference on Reverse Engineering*, pages 86–95. IEEE Computer Society Press :Los Alamitos, California.

Culwin F. MacLeod A. & Lancaster T. 2001. *Source code Plagiarism in UK HE Computing Schools, Issues, Attitudes and Tools, Technical Report No. SBU-CISM-01-02*. South Bank University.

Demeyer S, Sander T and Patrick S. 1999. " FAMIX 2.0 *The FAMOOS Information Exchange Model* ". http://seal.ifi.uzh.ch/fileadmin/User_Filemount/Vorlesungs_Folien/Informatik_II/SS06/famix20.pdf. diakses tanggal 08-10-2010 jam jam 22:30

Frakes W B and B. A. Nejmeh. 1984. *Software Reuse through Information Retrieval*. *SIGIR Forum* 21(1-2):30–36. British Computer Society Swinton : United Kingdom.

Freidig M. 1999. *Implementation of a prototype for generating XMI documents based on the FAMIX meta-model*. Institut für Informatik und angewandte Mathematik Universität Bern, Neubrückstr:Bern.

Goenamwan W, Ronald A., Krisantus S. 2005. *Penerapan Algoritma Edit Distance pada pendeteksian praktik plagiat*.Dept. Teknik Informatika Institut Teknologi: Bandung.

Ji J H, Woo G, Cho H G .2007. *A Source Code Linearization Teqnique for Detect Plagiarized Program*. Proc. Of the 12th ann. SIGCSE conf. On Innvtn.& tech. In comp.sci.edu (ItiCSE'07), SS : Courseware, pp 7377,07.

Kit S C, Wei L W, Kuok-Shoong W. 2007. *Simple Foward Method for Plagiarism Detection*. The Fifth International Conference onAdvances in Mobile Computing and Multimedia :Jakarta.

Lafore R. 1998. *Data Structures and Algorithms in Java*. Waite Group Press, Macmillan Computer Publishing.

LINFO (*The Linux Information Project*). 2004. http://www.linfo.org/source_code.html. diakses tanggal 12-10-2010, jam 0:25

Lin D. 1998. *Information-Theoretic Definition of Similarity*. Department of Computer Science University of Manitoba Winnipeg, Manitoba : Canada.

Petralli A C. 2008. *A Context Aware Algorithm to Extract Structural Changes Between FAMIX Models*. Software Evolution & Architecture Lab.Department of Informatics, University of Zurich.

Penyusun, Tim. 1989. *Kamus Besar Bahasa Indonesia*. Balai Pustaka:Jakarta

iParadigms teams. 2009. What is Plagiarism?. http://www.plagiarism.org/plag_article_what_is_plagiarism.html. Diakses tanggal 7 -10 -2010 jam 23:34.

PurboW Ono. 2009. http://opensource.telkomspeedy.com/wiki/index.php?title=Source_Code. diakses tanggal 08-10-2010 jam 00:17.

Sager. T, Abrahham B, Martin P, Christoph K. 2006. *Detecting Similar Java Classes Using Tree Algorithms*. Department of informatics University of Zurich, Switzerland.

Sheard J, Martin D, Selby M, Ian M, and Meaghan W. 2002. *Cheatingand plagiarism: perceptions and practices of first year it students.* In Proceedings of the 7thannual conference on Innovation and technology in computer science education, pages 183– 187. ACM Press.

Webster's Online Dictionary. *What is palgiarism*. http://www.websters-online-dictionary.org/ diakses tanggal 07-10-2010 jam jam 23:34.

LAMPIRAN

Daftar Source Code Uji Coba Program

I. Source Code I dan Percobaan A

```
public class StopWatch {
private long startTime = 0;
private long stopTime = 0;
private boolean running = false;
public void start() {
this.startTime = System.currentTimeMillis();
this.running = true;
}
public void stop() {
this.stopTime = System.currentTimeMillis();
this.running = false;
}
public long getElapsedTime() {
long elapsed;
if (running) {
elapsed = (System.currentTimeMillis() -
startTime);
}
else {
elapsed = (stopTime - startTime);
}
return elapsed;}
public long getElapsedTimeSecs() {
long elapsed;
if (running) {
elapsed = ((System.currentTimeMillis() -
startTime) / 1000);
}else {
elapsed = ((stopTime - startTime) / 1000);
}
return elapsed;}
public static void main(String[] args) {
StopWatch s = new StopWatch();
s.start();
s.stop();
System.out.println("elapsed time in
milliseconds: " + s.getElapsedTime());
} }
```

1.1 Percobaan B

```
public class StopWatch {
private long startTime = 0;
private long stopTime = 0;
private boolean running = false;
private int mulai=0; // Modifikasi
public StopWatch(){
            // Modifikasi
}
public void start() {
this.startTime =
System.currentTimeMillis();
this.running = true;}
public void stop() {
this.stopTime = System.currentTimeMillis();
this.running = false;}
public long getElapsedTime() {
long elapsed;
if (running) {
elapsed = (System.currentTimeMillis() -
startTime);
}
else {
   elapsed = (stopTime - startTime);
}
return elapsed;}
public long getElapsedTimeSecs() {
long elapsed;
if (running) {
elapsed = ((System.currentTimeMillis() -
startTime) / 1000);
}
else {
elapsed = ((stopTime - startTime) / 1000);
}
return elapsed;}
public static void main(String[] args) {
StopWatch s = new StopWatch();
s.start();
s.stop();
System.out.println("elapsed time in
milliseconds: " + s.getElapsedTime());
} }
```

1.2 Percobaan C

```
public class StopWatch {
private long startTime = 0;
private long stopTime = 0;
private boolean running = false;
public void start() {
this.startTime = System.currentTimeMillis();
this.running = true;
}
public void stop() {
this.stopTime = System.currentTimeMillis();
this.running = false;
}
public long getElapsedTime() {
long elapsed;
if (running) {
elapsed = (System.currentTimeMillis() -
startTime);}
else {
elapsed = (stopTime - startTime);
}
return elapsed;}
public long getElapsedTimeSecs() {
long elapsed;
if (running) {
elapsed = ((System.currentTimeMillis() -
startTime) / 1000);
}
else {
elapsed = ((stopTime - startTime) / 1000);
}
return elapsed;}
// Modifikasi Method
public void Modifikasi (){
}
public static void main(String[] args) {
StopWatch s = new StopWatch();
s.start();
s.Modifikasi();
//code you want to time goes here
s.stop();
System.out.println("elapsed time in
milliseconds: " + s.getElapsedTime());    } }
```

1.3 Percobaan D

```
public class StopWatchModifikasi {
private long startTime = 0;
private long stopTime = 0;
private boolean running = false;
public void mulai() {
this.startTime = System.currentTimeMillis();
this.running = true;
}
public void berhenti() {
this.stopTime = System.currentTimeMillis();
this.running = false;
}
public long getMilisecondTime() {
long elapsed;
if (running) {
elapsed = (System.currentTimeMillis() -
startTime);
}
else {
elapsed = (stopTime - startTime);
}
return elapsed;
}
public long getSecondTime() {
long elapsed;
if (running) {
elapsed = ((System.currentTimeMillis() -
startTime) / 1000);
}
else {
elapsed = ((stopTime - startTime) / 1000);
}
  return elapsed;
}
public static void main(String[] args) {
StopWatchModifikasi s = new
StopWatchModifikasi();
s.mulai();
s.berhenti();
System.out.println("elapsed time in
milliseconds: " + s.getMilisecondTime());
    }}
```

1.4 Percobaan E

```
package TestPackage;
import java.awt.event.AdjustmentListener;
import javax.swing.*;
import java.awt.*;
import javax.swing.JTree.*;
import java.awt.BorderLayout;
import java.awt.event.AdjustmentEvent;
import java.util.*;
import javax.swing.tree.DefaultMutableTreeNode;
import javax.swing.tree.MutableTreeNode;

public class Tree extends JFrame{
DefaultMutableTreeNode root,PNode,CNode,INode,
ConsNode; // Package , Class , InheritNode, dan
Construktor Node
JTree tree;
JScrollBar jsbver;
JScrollBar jsbhor;
public Tree(){
super("TREE GENERATION");
setSize(600, 600);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

public void initPrimer(Object p, Object ih){
root  = new DefaultMutableTreeNode("FAMIX
MODEL");
PNode = new DefaultMutableTreeNode(p);
INode = new DefaultMutableTreeNode(ih);
root.add(INode);
root.add(PNode);
}
public void initConstructor(String cons){
CNode.add(new DefaultMutableTreeNode(cons));
}
public void initGlobalVar(String [] gbl){
for (int i = 0; i < gbl.length; i++) {
if (!gbl[i].equals("null")){
CNode.add(new DefaultMutableTreeNode(gbl[i]));
}
```

```
}
}
public void initClass(String c){
CNode = new DefaultMutableTreeNode(c);
PNode.add(CNode);
}
public void initMethod (String [] met){
try{
for (int i = 0; i < met.length; i++) {
if (!met[i].equals("null")){
String [] mm = met[i].split("@");
CNode.add(new DefaultMutableTreeNode(mm[0]));
mm[1] = mm[1].replace("[", "").replace("]",
"");
String NN [] = mm[1].split(",");
for (int j = 0; j < NN.length; j++) {
if (j==0){
CNode.getLastLeaf().add(new
DefaultMutableTreeNode("PARAMETER_"+NN[0]));
 }
else{
DefaultMutableTreeNode bb = new
DefaultMutableTreeNode();
bb = (DefaultMutableTreeNode)
CNode.getLastLeaf().getParent();
bb.add(new
DefaultMutableTreeNode("PARAMETER_"+NN[j]));
}}
try {
DefaultMutableTreeNode dd = new
DefaultMutableTreeNode();
dd = (DefaultMutableTreeNode)
CNode.getLastLeaf().getParent();
dd.add(new DefaultMutableTreeNode(mm[2]));
} catch (Exception e) {
}
try {
DefaultMutableTreeNode dd2 = new
DefaultMutableTreeNode();
dd2 = (DefaultMutableTreeNode)
CNode.getLastLeaf().getParent();
dd2.add(new DefaultMutableTreeNode(mm[3]));
} catch (Exception e) {
```

```
}          }   }
}catch(Exception ex){
}   }
public void bulid(){
tree = new JTree(root);
getContentPane().add(tree,
BorderLayout.CENTER);
JScrollPane scrollpane = new JScrollPane(tree);
getContentPane().add(scrollpane, "Center");
}
public JTree getTree(){
    return this.tree;
} }
```

1.5 Percobaan F

Source code I

```
class StopWatch {
private long startTime = 0;
private long stopTime = 0;
private boolean running = false;
public void start() {
this.startTime = System.currentTimeMillis();
this.running = true;
}
public void stop() {
this.stopTime = System.currentTimeMillis();
this.running = false;
}
public long getElapsedTime() {
long elapsed;
if (running) {
elapsed = (System.currentTimeMillis() -
startTime);
}
else {
elapsed = (stopTime - startTime);
}
return elapsed;}
public long getElapsedTimeSecs() {
long elapsed1;
if (running) {
elapsed1 = ((System.currentTimeMillis() -
```

```
startTime) / 1000);
}
else {
elapsed1 = ((stopTime - startTime) / 1000);
}
return elapsed1;
}
public double ratawhile()
{
int [] array1 = {1,2,3,4,5};
double x = 0;
int i=0, sum =0;
while(i<array1.length){
sum += array1[i];
i += 1;
}
System.out.println(sum);
x = sum / (double) array1.length;
return x;
}
public static void main(String[] args) {
StopWatch s = new StopWatch();
s.start();
s.stop();
System.out.println("elapsed time in
milliseconds: " + s.getElapsedTime());
}
}
```

Source code II

```
class StopWatch {
private long startTime = 0;
private long stopTime = 0;
private boolean running = false;
public void start() {
this.startTime = System.currentTimeMillis();
this.running = true;
}
public void stop() {
this.stopTime = System.currentTimeMillis();
this.running = false;
}
```

```
public long getElapsedTime() {
long elapsed;
if (running) {
 elapsed = (System.currentTimeMillis() -
startTime);}
else { elapsed = (stopTime - startTime);
} return elapsed;}
public long getElapsedTimeSecs() {
long elapsed1;
if (running) {
elapsed1 = ((System.currentTimeMillis() -
startTime) / 1000);
}
else {
elapsed1 = ((stopTime - startTime) / 1000);
}
return elapsed1;
}
public double ratafor(){
int [] array1 = {1,2,3,4,5};
double ret = 0;
int sum = 0;
for(int i = 0 ; i< array1.length;i++)
{
sum += array1[i];
}
System.out.println(sum);
ret = sum / (double) array1.length;
return ret;
}
public static void main(String[] args) {
StopWatch s = new StopWatch();
s.start();
s.stop();
System.out.println("elapsed time in
milliseconds: " + s.getElapsedTime());
}
}
```