BAB III

ANALISIS DAN PERANCANGAN SISTEM

3.1 Kondisi Saat Ini

Salah satu hal yang membuat praktik plagiarisme dapat dilakukan dengan mudah adalah kemajuan dalam bidang teknologi dan informasi. Informasi apapun di seluruh dunia dapat diakses oleh siapa saja, kapan saja, dan dimana saja. Selain itu, kemajuan dalam bidang teknologi, seperti perangkat *flashdisk* yang kecil dan mudah dibawa-bawa mempermudah siapapun untuk menyimpan bahkan saling bertukar informasi/data dengan orang lain.

Secara khusus dalam cakupan mahasiswa fakultas ICT (*Information and Communication Technology*) di Universitas Multimedia Nusantara untuk mata kuliah Algoritma dan Pemrograman, praktik plagiarisme cukup banyak ditemukan dalam bentuk penyalinan kode program. Sebanyak 25-30 kasus plagiarisme kode program ditemukan secara rata-rata untuk tiap kelasnya dalam 1 semester.

Berdasarkan hasil wawancara yang dilakukan terhadap beberapa dosen dan asisten laboratorium mata kuliah Algoritma dan Pemrograman, konsep kelas paralel menjadi salah satu masalah tersendiri dalam proses mengindentifikasi kecenderungan plagiarisme kode program. Dimana, dengan konsep kelas paralel terdapat kemungkinan dosen mata kuliah Algoritma dan Pemrograman yang mengajar di setiap kelas berbeda satu sama lain. Hal ini memungkinkan mahasiswa dari kelas X yang diajar oleh dosen A menyalin kode program milik

mahasiswa dari kelas Y dengan dosen B. Dengan demikian, dosen A tentu tidak akan menyadari tindak plagiarisme yang dilakukan oleh mahasiswa X karena beliau tidak melakukan pengecekan/pendeteksian dengan menggunakan dokumen kode program milik dosen B.

Selain itu, juga terdapat kemungkinan adanya mahasiswa yang melakukan teknik *edit* setelah teknik *copy-paste* kode program milik temannya. Teknik *edit* ini ditujukan agar hasil pendeteksian kecenderungan plagiarisme yang dihasilkan lebih kecil persentasenya (tidak 100%). Cara *copy-paste-edit* ini dapat dengan mudah dilakukan karena sekarang proses pengujian program dapat dilakukan dengan mudah. Jika setelah di-*edit*, hasil keluaran (*output*) tetap sesuai dengan apa yang diharapkan/diminta oleh dosen maka terdapat kemungkinan dosen yang melakukan pengecekan/pendeteksian atas dokumen kode program yang dibuat oleh mahasiswa tersebut tidak menyadari tindak plagiarisme tersebut.

Beberapa bentuk manipulasi kode program yang seringkali dilakukan mahasiswa berdasarkan hasil wawancara yang dilakukan, yaitu: mengubah komentar (5 suara), mengubah *string output* program (6 suara), mengubah nama variabel (6 suara), mengubah nama fungsi dan/atau parameter fungsi (4 suara), mengubah bentuk *for* menjadi *while* atau sebaliknya (3 suara), mengubah *conditional statement* (2 suara), dan menukar posisi baris kode program (4 suara).

3.2 Masalah yang Dihadapi

Berdasarkan kondisi saat ini di Universitas Multimedia Nusantara, ditemukan beberapa masalah pokok dalam tindak plagiarisme kode program, yaitu.

1. Banyaknya tugas pemrograman yang dibuat oleh mahasiswa, yang harus dideteksi bobot kecenderungan plagiarismenya oleh tenaga pengajar (dosen).
2. Adanya konsep kelas paralel yang memungkinkan mahasiswa menyalin kode program milik mahasiswa kelas lainnya sehingga tidak terdeteksi oleh dosen di kelas masing-masing.
3. Sulitnya melakukan dokumentasi (pencatatan) hasil pengecekan/ pendeteksian untuk dokumen kode program dalam jumlah yang cukup banyak.
4. Adanya teknik *editing* yang memungkinkan mahasiswa melakukan perubahan kecil setelah tindak plagiat (*copy-paste*) sehingga dosen tidak menyadarinya ketika melakukan pemeriksaan/pendeteksian.

3.3 Pemecahan Masalah

Berdasarkan masalah yang ditemukan tersebut, ditawarkan solusi berupa pembuatan sebuah aplikasi pendeteksi plagiarisme yang memiliki fitur-fitur yaitu.

1. Pendeteksi plagiarisme antara dua dokumen kode program dalam satu waktu proses.

2. Menampilkan bobot persentase kecenderungan plagiarisme antara dua dokumen kode program dengan algoritma *Levenshtein Distance* (metode pencocokan *by character*) dan algoritma *Brute Force* (metode pencocokan *by line*).
3. Pendeteksi plagiarisme yang mampu memproses seluruh dokumen kode program dalam sebuah *folder* secara otomatis.
4. Menghasilkan bobot persentase kecenderungan plagiarisme antara setiap dua dokumen kode program yang terdapat di dalam *folder* yang dipilih dengan algoritma *Levenshtein Distance* (metode pencocokan *by character*) dan algoritma *Brute Force* (metode pencocokan *by line*).
5. Menyimpan hasil bobot prosentase kecenderungan plagiarisme sebagaimana yang dimaksud pada fitur no (4) dalam bentuk dokumen teks (.txt).
6. Adanya teknik *preprocessing* yang memungkinkan aplikasi dapat mendeteksi dua dokumen kode program atau lebih yang telah di-*edit* secara lebih cermat dan menghasilkan bobot persentase yang lebih akurat.

3.4 Masukkan dan Keluaran Sistem

Masukkan yang dibutuhkan sistem berupa.

1. Lokasi penyimpanan dari dua dokumen kode program yang akan dibandingkan.

2. Lokasi penyimpanan dari satu *folder* yang berisi sekurang-kurangnya dua dokumen kode program yang akan dan dapat dibandingkan.

Keluaran dari sistem berupa.

1. Bobot persentase kecenderungan plagiarisme dengan algoritma *Levenshtein Distance* dan *Brute Force* antara dua dokumen kode program yang dibandingkan.
2. Sebuah *file* teks dengan nama "CPlag_Result" dalam format .txt yang berisi hasil pencocokan antara setiap dua dokumen kode program yang dibandingkan dalam satu *folder* yang dipilih.

3.5 Perancangan Sistem

Sistem ini dirancang dengan model spiral, dimana perancangan dimulai dari fitur paling dasar, kemudian secara progresif dikembangkan fitur-fitur lain, sehingga dihasilkan sistem secara utuh.

Tahapan perancangan dari tiap putaran proses spiral yang dilakukan terbagi menjadi lima bagian utama, yaitu.

1. Proses *preprocessing*.
2. Penghitungan bobot persentase kecenderungan plagiarisme dengan algoritma *Levenshtein Distance*.
3. Penghitungan bobot persentase kecenderungan plagiarisme dengan algoritma *Brute Force*.

4. Perbandingan antara dua dokumen kode program.
5. Perbandingan antara seluruh dokumen kode program di dalam *folder*.

Selain kelima bagian utama tersebut, juga dilakukan perancangan *data flow diagram*, *pseudocode* proses, struktur hirarki menu, dan tampilan antarmuka.

3.5.1 Proses Sistem

Sebagaimana yang telah disebutkan pada subbab 3.5, pada bagian ini akan dijelaskan secara lebih detail tentang keenam bagian utama yang menjadi proses inti dari aplikasi pendeteksi plagiarisme kode program yang dibuat.

A. Proses Preprocessing

Pada tahap *preprocessing*, dokumen teks kode program bahasa C yang akan dibandingkan akan diubah menjadi suatu bentuk yang sama/seragam, dengan tujuan agar bobot persentase kecenderungan plagiarisme yang dihasilkan dapat lebih tinggi. Sebagaimana yang diungkapkan oleh Novanta (2009, p.26), teknik ini diperlukan karena algoritma pencocokan *string* yang umumnya digunakan dalam pendeteksian plagiarisme hanya membandingkan secara eksplisit kedua *string* yang dibandingkan tanpa memperhatikan unsur-unsur lain yang menjadi karakteristik *string* tersebut.

Teknik *preprocessing* akan mengubah data mentah (dokumen kode program awal) menjadi sebuah data yang siap diproses ke tahap selanjutnya (kode

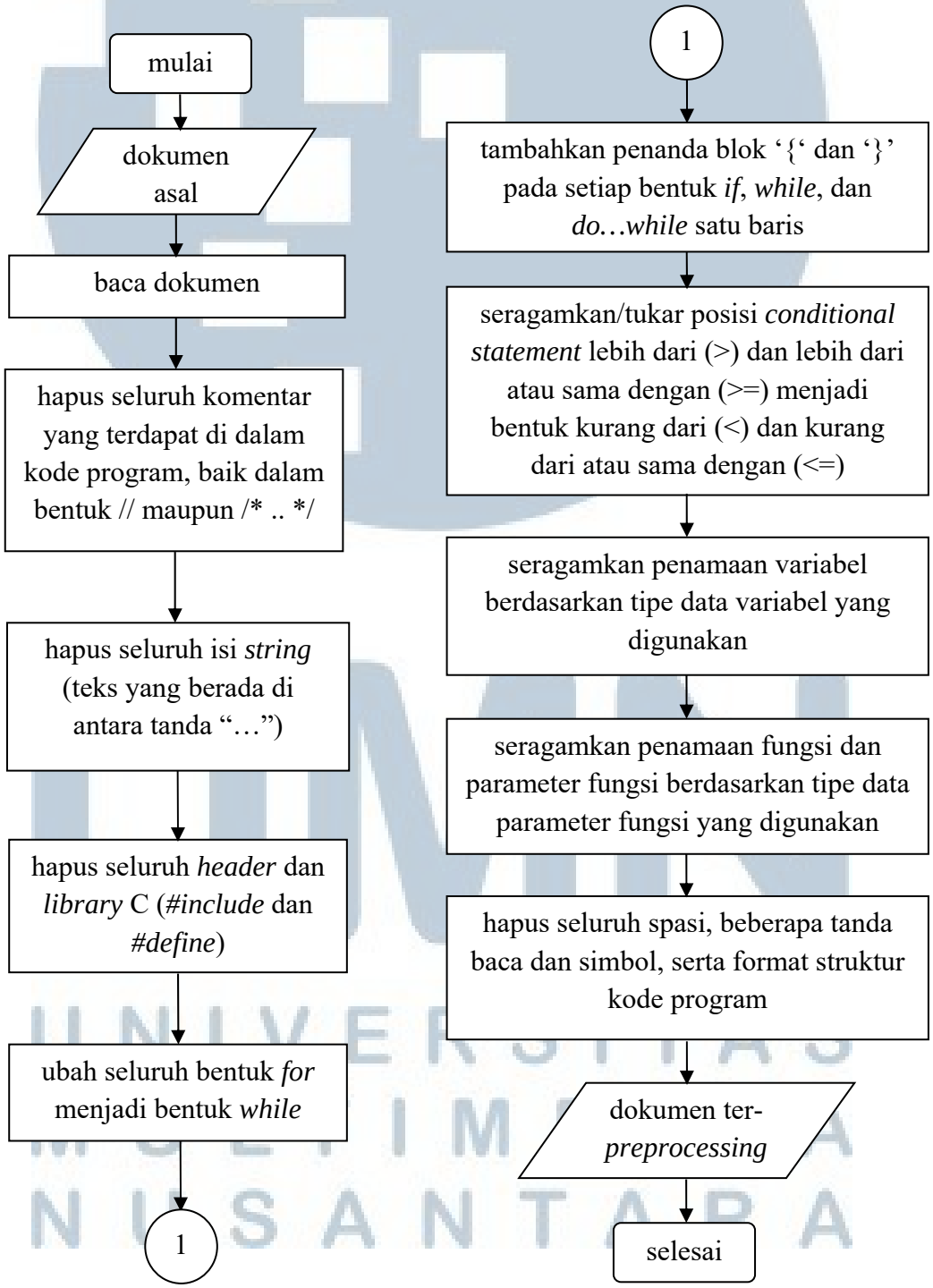
program yang sudah di format). Selain berguna dalam pendeteksian plagiarisme dokumen teks biasa, teknik ini juga sangat berguna dalam pendeteksian plagiarisme dokumen kode program. Hal ini disebabkan karena dalam proses plagiarisme kode program ada beberapa bagian yang dapat diubah tanpa mengubah makna kode program secara keseluruhan.

Proses *preprocessing* yang terjadi di dalam aplikasi pendeteksi plagiarisme kode program terdiri dari 9 tahap, yaitu.

1. Menghapus seluruh komentar yang terdapat di dalam kode program, baik dalam bentuk // maupun /* .. */.
2. Menghapus seluruh isi *string* (teks yang berada di antara tanda "…").
3. Menghapus seluruh *header* dan *library* C (*#include* dan *#define*).
4. Mengubah seluruh bentuk *for* menjadi bentuk *while*.
5. Menambahkan penanda blok '{' dan '}' pada setiap bentuk *if*, *while*, dan *do…while* satu baris.
6. Menyeragamkan/menukar posisi *conditional statement* lebih dari (>) dan lebih dari atau sama dengan (>=) menjadi bentuk kurang dari (<) dan kurang dari atau sama dengan (<=).
7. Menyeragamkan penamaan variabel berdasarkan tipe data variabel yang digunakan.
8. Menyeragamkan penamaan fungsi dan parameter fungsi berdasarkan tipe data parameter fungsi yang digunakan.
9. Menghapus seluruh spasi, beberapa tanda baca dan simbol, serta memformat struktur kode program.

Jika digambarkan dalam bentuk *flowchart*, maka tahapan dari proses *preprocessing* yang terjadi dapat dilihat pada Gambar 3.1.

Gambar 3.1 *Flowchart* Proses *Preprocessing*



B. Penghitungan Bobot Persentase Kecenderungan Plagiarisme dengan Algoritma Levenshtein Distance

Setelah *preprocessing* dilakukan, proses selanjutnya yang dilakukan adalah pembobotan nilai untuk menentukan persentase kecenderungan plagiarisme antara dokumen kode program yang dibandingkan. Metode pertama yang digunakan dalam pembobotan adalah dengan menggunakan algoritma *Levenshtein Distance*.

Algoritma *Levenshtein Distance* digunakan untuk membandingkan dokumen kode program secara karakter per karakter (*by character*). Hal ini sesuai dengan fungsionalitas dari *Levenshtein Distance* yang mampu menghitung jarak (perbedaan) antara dua *sequence* (*string*) berdasarkan operasi minimum yang diperlukan untuk merubah satu *string* menjadi *string* lainnya, melalui teknik *substitution*, *insertion*, dan *deletion*.

Secara umum, dalam prosesnya kedua *string* akan dimasukkan kedalam sebuah variabel *array* dimensi dua dengan ukuran sebesar panjang dari *string* 1 dan dari panjang *string* 2 (arr[*length string* 1, *length string* 2]). Selanjutnya baris dan kolom ke 0 akan diisi secara inkremen sesuai panjang dimensi *array*. Sedangkan untuk isi dari sel dimensi *array* lainnya (arr[i, j]) diisi dengan nilai terkecil antara:

1. Nilai pada sel arr[i-1, j] + 1, atau
2. Nilai pada sel arr[i, j-1] + 1, atau
3. Nilai pada sel arr[i-1, j-1] + *cost*.

Nilai dari *cost* sendiri didapatkan dari perbandingan karakter yang terdapat di dalam *array* pada posisi i, j. jika karakter pada baris i sama dengan karakter pada kolom j atau sebaliknya, maka nilai *cost* adalah 0. Sedangkan jika karakter pada baris i berbeda dengan karakter pada kolom j atau sebaliknya, maka nilai *cost* adalah 1.

Secara garis besar, algoritma *Levenshtein Distance* yang akan diimplementasikan pada sistem dapat dituliskan sebagai berikut:

Algoritma 3.1 *Levenshtein Distance*

```
1. s = source document;
2. c = copy document;
3. m = s.length;
4. n = c.length;
5. max_line = max(s.length , c.length);
6. arr[0, 0] = 0;
7. for  i = 0 to m
8. arr[i, 0] = i;
9. for  j = 0 to n
10. arr[0, j] = j;
11. for  i = 1 to m
12. for  j = 1 to n
13. if  s[i] == c[j] then cost = 0;
14. else  cost = 1;
15. arr[i, j] = min(arr[i-1, j] + 1, arr[i, j-1] + 1, arr[i-
    1, j-1] + cost)
16. return arr[m, n];
```

Setelah *array* dua dimensi tersebut terisi semua, maka akan didapat jarak minimum yang dibutuhkan untuk mengubah dari *string* 1 menjadi *string* 2 atau dalam bahasa lainnya adalah jumlah karakter yang berbeda antara kedua *string* tersebut. Dengan demikian, sistem akan dapat menghitung bobot persentase kecenderungan plagiarisme antara kedua *string* dokumen kode program tersebut dengan rumus (Goenawan, 2005, p.3):

$$Lev = 100\% - \frac{\text{nilai sel arr[i, j]}}{\text{max}} * 100\% \quad (1)$$

**max = jumlah karakter terpanjang antara string Kode Program 1 dan string Kode Program 2*

Kompleksitas dari algoritma *Levenshtein Distance* sendiri adalah O(mn), dimana 'm' adalah panjang karakter dari *string* 1 dan 'n' adalah panjang karakter dari *string* 2.

C. Penghitungan Bobot Persentase Kecenderungan Plagiarisme dengan Algoritma Brute Force

Selain menggunakan algoritma *Levenshtein Distance*, sistem juga melakukan penghitungan bobot persentase kecenderungan plagiarisme dokumen kode program dengan menggunakan algoritma *Brute Force*. Hal ini dilakukan untuk menutupi kekurangan dari algoritma *Levenshtein Distance*, dimana algoritma *Levenshtein Distance* dapat 'tertipu' ketika membandingkan dokumen kode program yang barisnya ditukar satu sama lain. Sebagai contoh:

Kode Program 1:	**Kode Program 2:**
int a, b, c; char x, y, z; double m, n, o;	double m, n, o; int a, b, c; char x, y, z;

Gambar 3.2 Contoh Kode Program

Jika kedua kode program tersebut dibandingkan dengan menggunakan algoritma *Levenshtein Distance*, maka secara kasar yang akan terdeteksi hanya baris yang berisi *string* 'double m, n, o'. Hal ini disebabkan, secara konsep, algoritma *Levenshtein Distance* akan menghitung langkah minimum yang diperlukan untuk mengubah dari *string* pada Kode Program 1 menjadi *string* pada Kode Program 2. Sehingga proses yang terjadi adalah penghapusan baris pertama (int a, b, c;) dan kedua (char x, y, z;) lalu penyisipan (*insert*) kedua baris tersebut setelah baris ketiga.

Kode Program 1:	**Kode Program 2:**
int a, b, c; (*delete*)	
char x, y, z; (*delete*)	
double m, n, o;	double m, n, o;
	int a, b, c; (*insert*)
	char x, y, z; (*insert*)

Gambar 3.3 Ilustrasi Proses Algoritma *Levenshtein Distance*

Untuk menutupi kekurangan inilah, digunakan algoritma *Brute Force* yang memakai konsep pencocokan baris per baris (*by line*). Pada algoritma *Brute Force*, setiap baris pada dokumen kode program yang memiliki jumlah baris terpendek akan dicocokan dengan seluruh baris pada dokumen kode program lainnya. Jika ditemukan kecocokan, maka indikasi kecocokan akan bertambah dan baris tersebut akan dihilangkan/dihapus. Hal ini dilakukan agar, jika terdapat 2 baris kode program yang identik pada Kode Program 1, sedangkan hanya terdapat

1 baris kode program yang identik dengan kedua baris kode pada Kode Program 2, maka indikasi kecocokan hanya akan bertambah 1.

Secara garis besar, algoritma *Brute Force* yang akan diimplementasikan pada sistem dapat dituliskan sebagai berikut (diasumsikan panjang baris dokumen 1 lebih pendek dibanding panjang baris dokumen 2).

Algoritma 3.2 *Brute Force*

```
1. s = source document;
2. c = copy document;
3. min_line = s.line.lenght;
4. max_line = c.line.lenght;
5. for  i = 0 to min_line
6. for  j = 0 to max_line
7. if  s.line[i] == c.line[j] then
8. sama++;
9. remove(c.line[j]);
```

Lalu untuk menghasilkan bobot persentase kecenderungan plagiarisme berdasarkan algoritma *Brute Force* dapat dilakukan dengan teknik statistik sederhana (penghitungan persentase), dengan rumus (Warsito, 1992, p.59):

$$Brute = \frac{\text{Jumlah baris yang sama}}{\text{Max}} * 100\% \qquad (2)$$

*max = jumlah baris terpanjang antara Kode Program 1 dan Kode Program 2

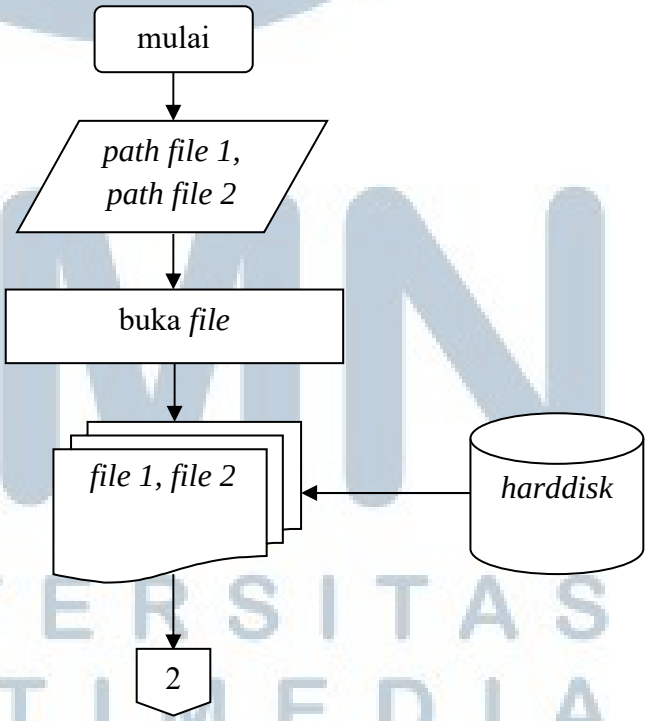
Kompleksitas dari algoritma *Brute Force* sendiri adalah untuk kasus *worse case* adalah O(mn), dimana 'm' adalah jumlah baris dari Kode Program 1 dan 'n' adalah jumlah baris dari Kode Program 2. Sedangkan untuk kasus *best case* adalah O(x), dimana x adalah jumlah baris terpanjang antara Kode Program 1 dan Kode Program 2.

D. Perbandingan Antara Dua Dokumen Kode Program

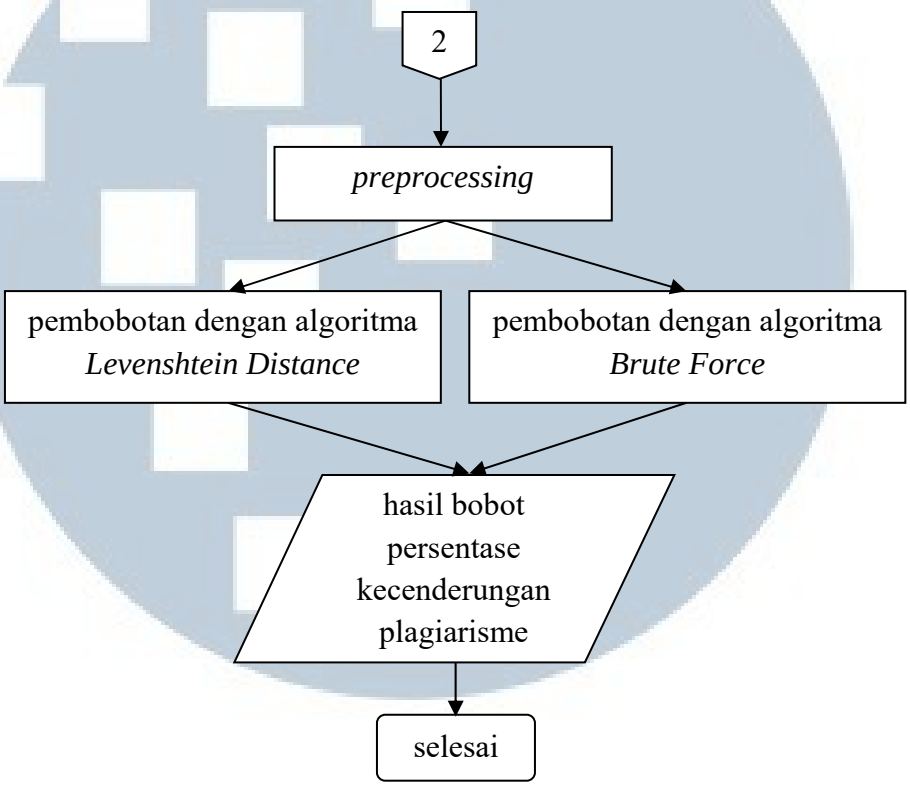
Setelah ketiga proses di atas selesai dirancang, tahapan berikutnya adalah merancang sebuah fitur yang dapat mengintegrasikan ketiga proses tersebut sehingga dapat menghasilkan bobot persentase kecenderungan plagiarisme sebagaimana yang diharapkan.

Pada fitur ini, *user* hanya dapat membandingkan dua dokumen kode program dalam satu waktu proses. Dimana, isi dari setiap dokumen kode program yang dibandingkan akan ditampilkan pada layar. Alur dari proses perbandingan 2 dokumen kode program dapat dilihat pada Gambar 3.4 di bawah ini.

Gambar 3.4 *Flowchart* Proses Perbandingan Antara Dua Dokumen Kode Program



Gambar 3.4 *Flowchart* Proses Perbandingan Antara Dua Dokumen Kode Program (Lanjutan)

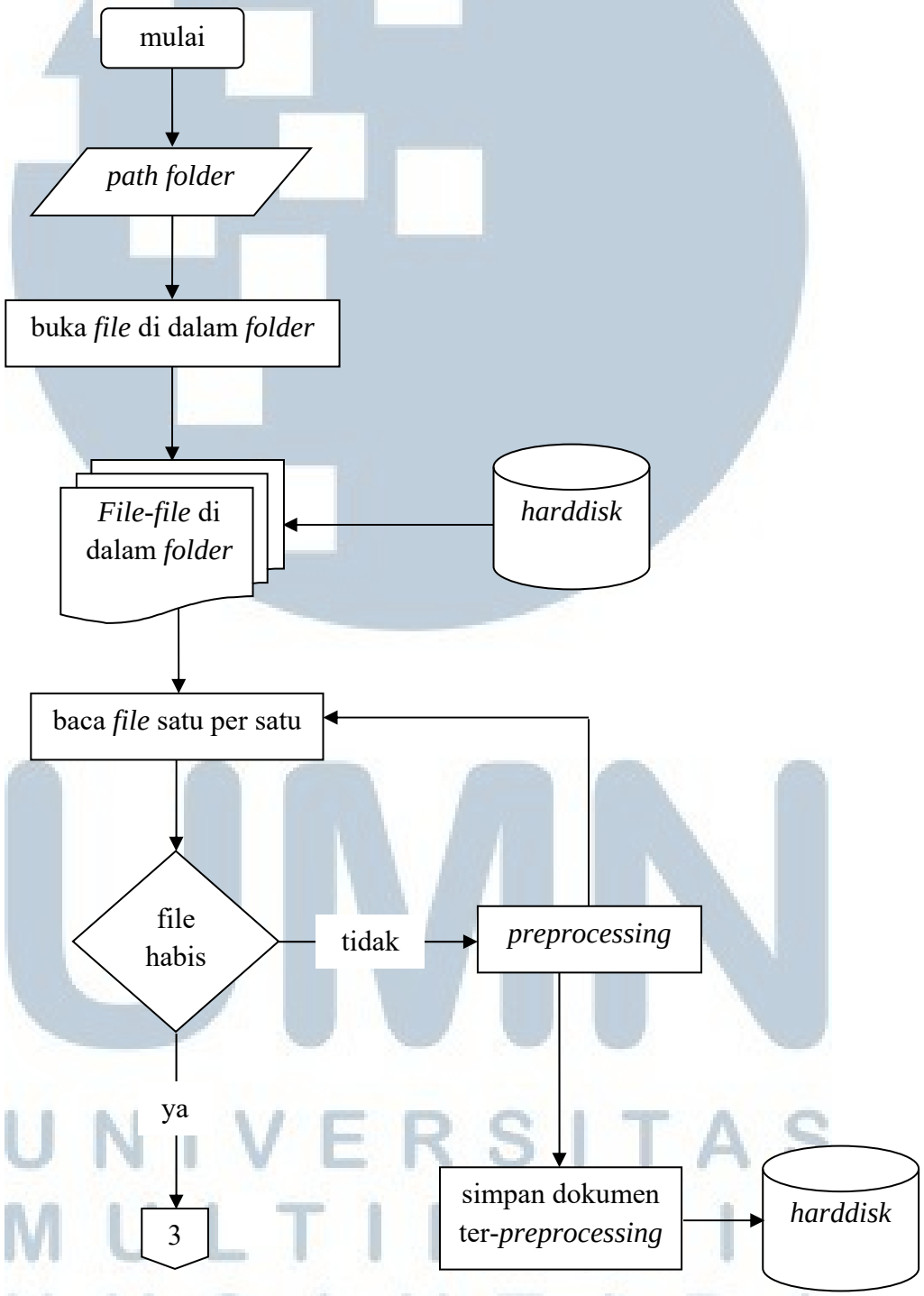


E. Perbandingan Seluruh Dokumen Kode Program di dalam Folder

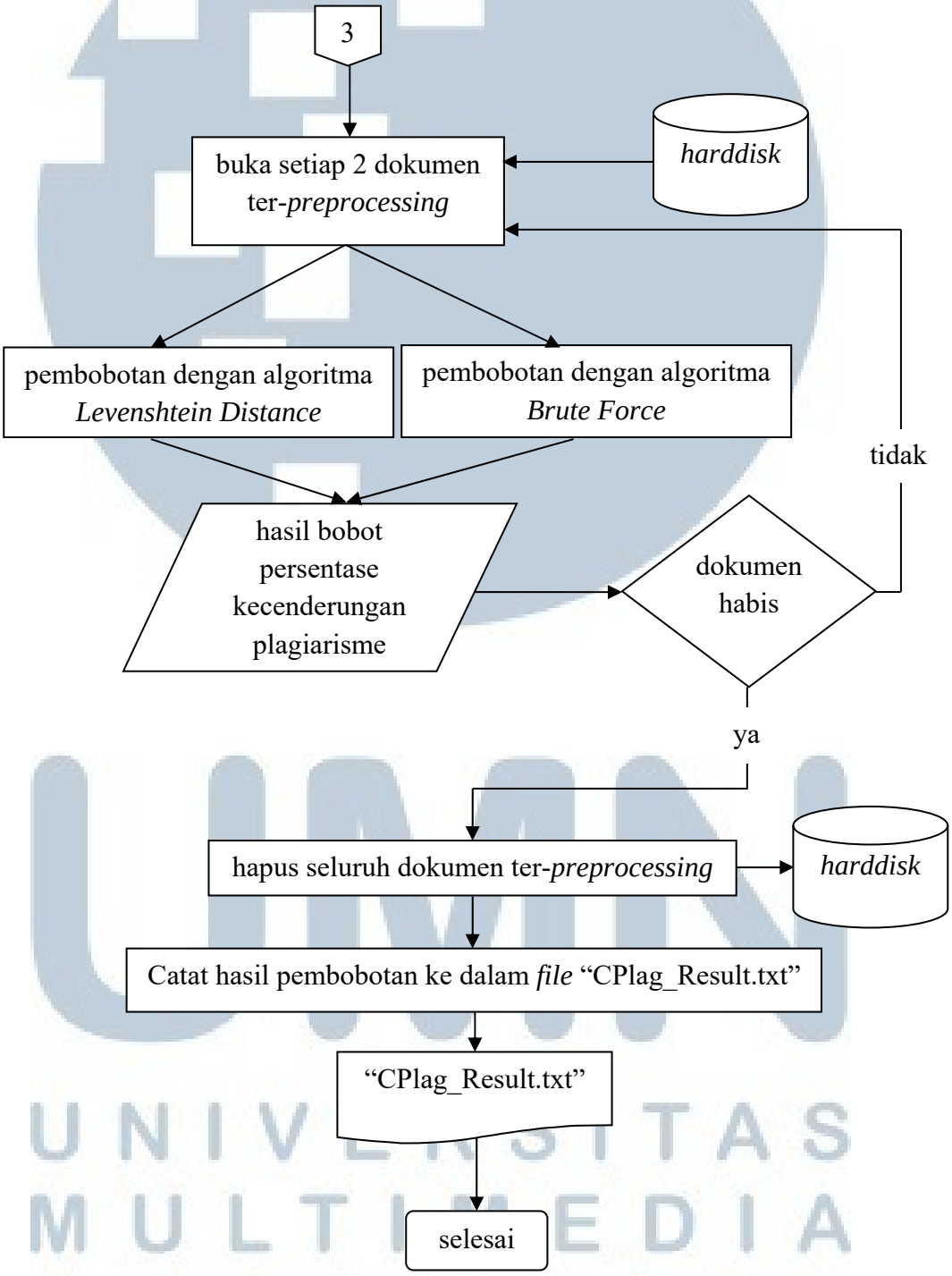
Selain fitur perbandingan antara dua dokumen kode program, juga terdapat fitur perbandingan seluruh dokumen kode program di dalam *folder*. Pada fitur ini, *user* dapat mendapatkan bobot persentase kecenderungan plagiarisme antara setiap dua dokumen kode program yang terdapat di dalam *folder* tersebut hanya dalam satu kali proses.

Alur dari proses perbandingan seluruh dokumen kode program di dalam *folder* dapat dilihat pada Gambar 3.5.

Gambar 3.5 *Flowchart* Proses Perbandingan Antara Seluruh Dokumen Kode Program di dalam *Folder*

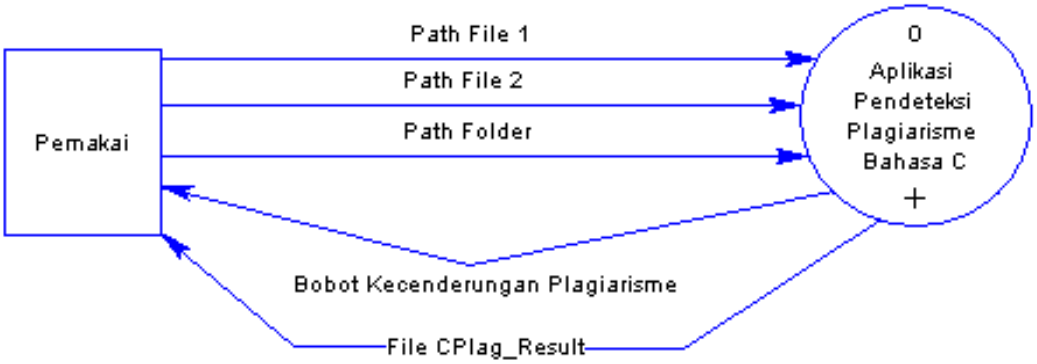


Gambar 3.5 *Flowchart* Proses Perbandingan Antara Seluruh Dokumen Kode Program di dalam *Folder* (Lanjutan)

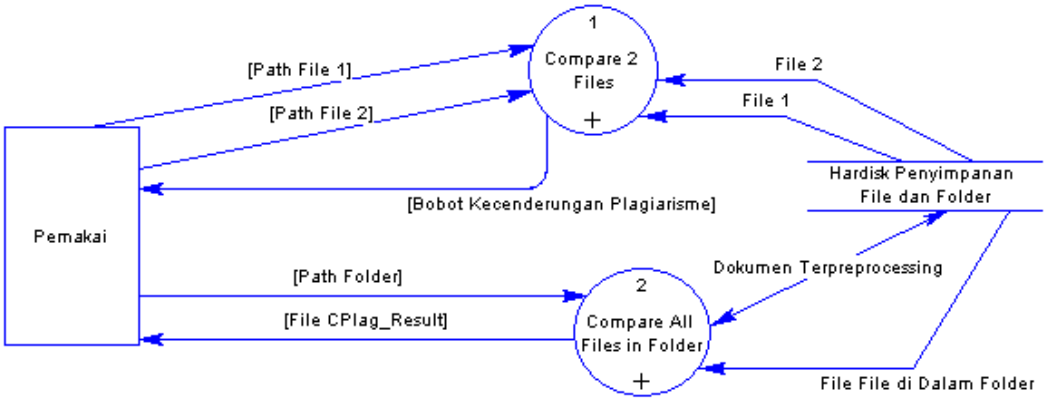


3.5.2 Data Flow Diagram

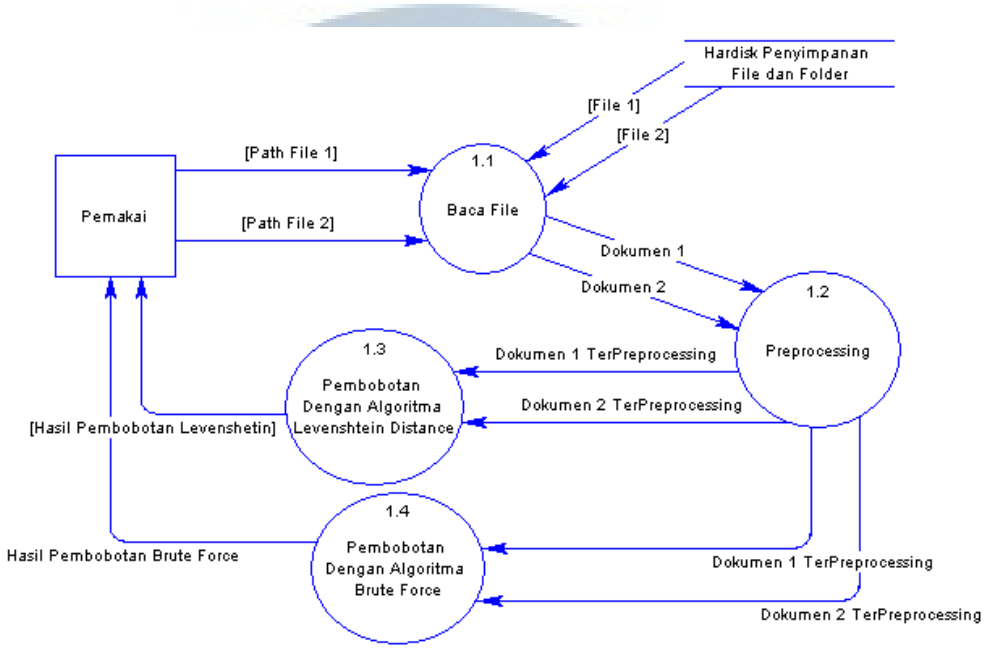
Berdasarkan data input-output dan proses yang terjadi di dalam sistem, maka dapat digambarkan *data flow diagram* sebagai berikut.



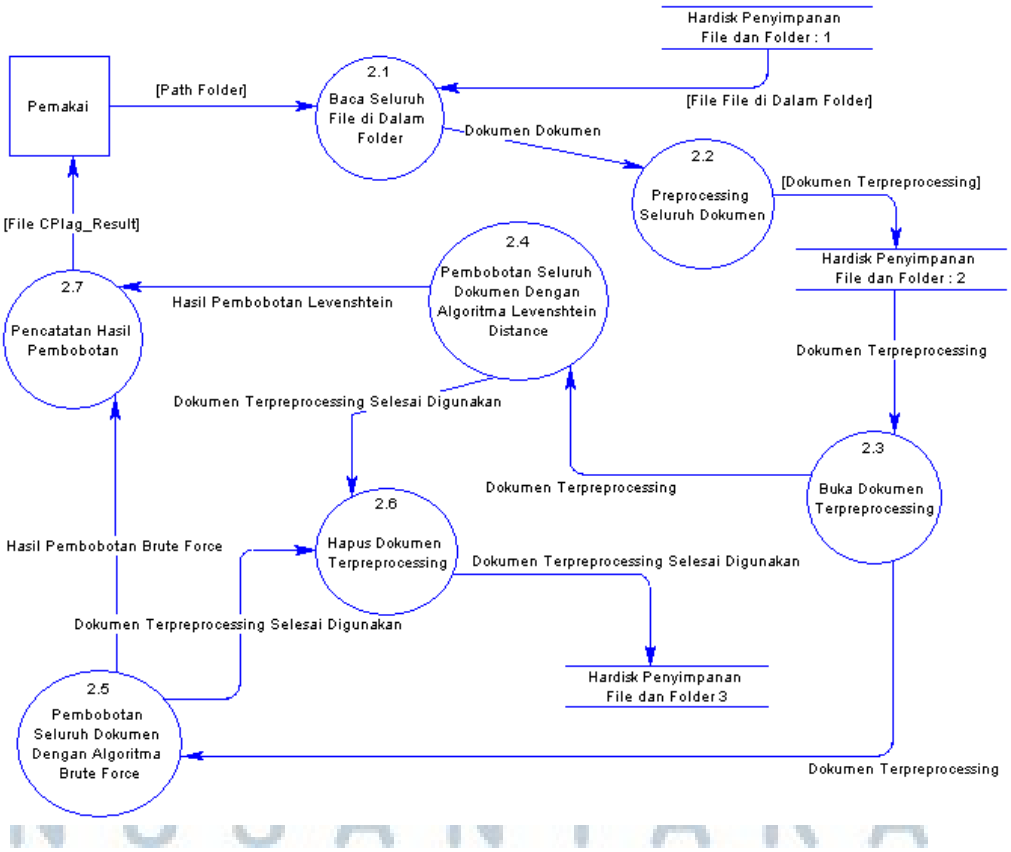
Gambar 3.6 *Context Diagram*



Gambar 3.7 *Data Flow Diagram Level* 1



Gambar 3.8 *Data Flow Diagram Level* 2 Subproses "*Compare 2 Files*"



Gambar 3.9 *Data Flow Diagram Level* 2 Subproses "*Compare All Files in Folder*"

3.5.3 Pseudocode Proses

Berdasarkan *data flow diagram* yang terdapat pada subbab 3.5.2, maka proses yang terjadi di dalamnya dapat digambarkan melalui *pseudocode* proses di bawah ini.

Algoritma 3.3 Proses Baca *File*

```
1. fs1= FileStream(File1, Open);
2. sr1 = StreamReader(fs1);
3. txtFile1 = sr1.ReadToEnd();
4. fs2= FileStream(File2, Open);
5. sr2 = StreamReader(fs2);
6. txtFile2 = sr2.ReadToEnd();
```

Algoritma 3.4 Proses *Preprocessing*

```
1. try
2. preprocessing(txtFile1);
3. preprocessing(txtFile2);
4. catch
5. MessageBox.Show("Error Message – There is Syntax Error");
```

Algoritma 3.5 Proses Pembobotan Dengan Algoritma *Levenshtein Distance*

```
1. result_Lev = Levenshtein_Distance(txtFile1, txtFile2);
2. txtResult = "Lev : " + result_Lev;
```

Algoritma 3.6 Proses Pembobotan Dengan Algoritma *Brute Force*

```
1. result_Brute = Brute_Force(txtFile1, txtFile2);
2. txtResult = "Brute : " + result_Brute;
```

Algoritma 3.7 Proses Baca Seluruh *File* Di Dalam *Folder*

```
1. path_file_c[] = Directory.GetFiles(txtFolder, "*.c");
2. path_file_cpp[] = Directory.GetFiles(txtFolder, "*.cpp");
3. path_file[] = path_file_c.Union(path_file_cpp).ToArray;
```

Algoritma 3.8 Proses *Preprocessing* Seluruh Dokumen

```
1. jumlah_file = path_file.Count();
2. System.IO.Directory.CreateDirectory(txtPreFolder);
3. if  jumlah_file >= 2
4. for i = 0 to jumlah_file
5. txtPre = preprocessing(file(i));
6. System.IO.File.WriteAllText(txtPreFolder+path file(i),
   txtPre);
```

Algoritma 3.9 Proses Buka Dokumen Ter-*preprocessing*

```
1. path_file_c[] = Directory.GetFiles(txtPreFolder, "*.c");
2. path_file_cpp[] = Directory.GetFiles(txtPreFolder,
   "*.cpp");
3. path_file[] = path_file_c.Union(path_file_cpp).ToArray;
```

Algoritma 3.10 Proses Pembobotan Seluruh Dokumen Dengan Algoritma *Levenshtein Distance*

```
1. for i = 0 to jumlah_file
2. for j = i + 1 to jumlah_file
3. result_Lev = Levenshtein_Distance(file(i), file(j));
4. txtResult += "Result = Lev : " + result_Lev;
```

Algoritma 3.11 Proses Pembobotan Seluruh Dokumen Dengan Algoritma *Brute Force*

```
1. for i = 0 to jumlah_file
2. for j = i + 1 to jumlah_file
3. result_Brute = Brute_Force(file(i), file(j));
4. txtResult += "Result = Brute : " + result_Brute;
```

Algoritma 3.12 Proses Hapus Dokumen Ter-*preprocessing*

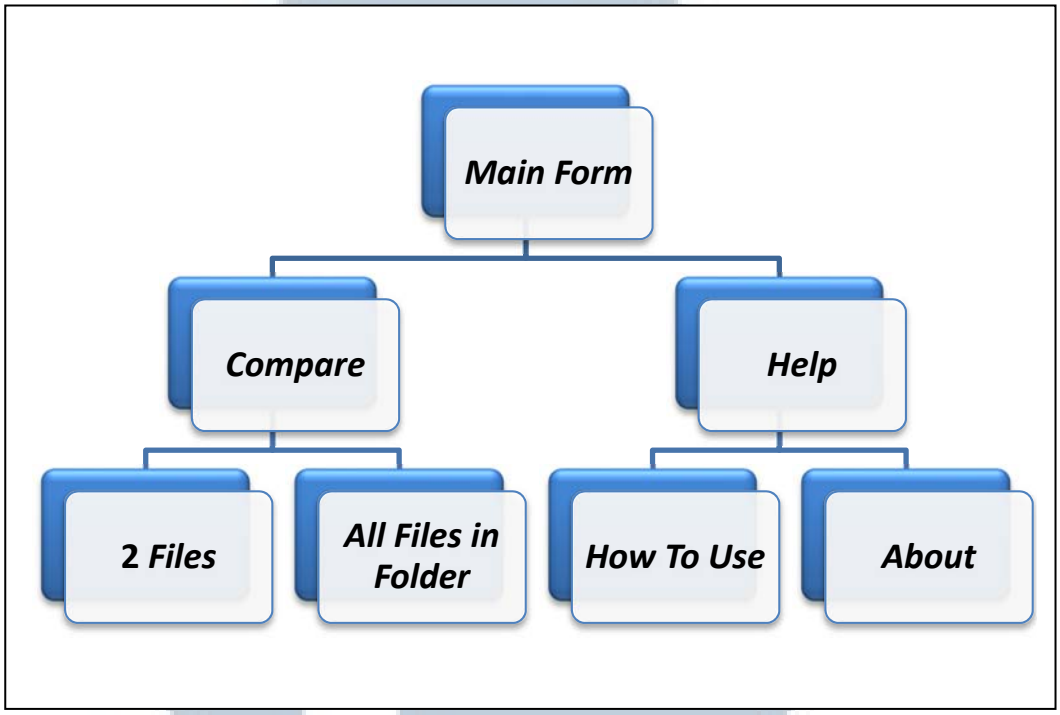
```
1. System.IO.Directory.Delete(txtPreFolder);
```

Algoritma 3.13 Proses Pencatatan Hasil Pembobotan

```
1. System.IO.File.WriteAllText("CPlag_Result.txt",
   txtResult);
```

3.5.4 Struktur Hirarki Menu

Struktur hirarki menu dari aplikasi pendeteksi plagiarisme yang dibuat dapat dilihat pada Gambar 3.8 (Barfield, 2004).



Gambar 3.10 Struktur Hirarki Menu

Dari menu utama, aplikasi dibagi menjadi 2 bagian besar, yakni "*Compare*" dan "*Help*". Menu "*Compare*" dibagi lagi menjadi 2 bagian, yakni "*Compare 2 Files*" dan "*Compare All Files in Folder*". Halaman "*Compare 2 Files*" berfungsi untuk membandingkan 2 buah dokumen kode program dalam satu waktu proses. Sedangkan halaman "*Compare All Files in Folder*" berfungsi untuk membandingkan seluruh dokumen kode program yang terdapat dalam sebuah *folder* yang dipilih user.

Pada menu "*Help*", terdapat navigasi untuk mengakses halaman "*How To Use*" dan "About". Halaman "*How To Use*" berfungsi untuk menjelaskan fungsionalitas dari aplikasi pendeteksi plagiarisme kepada user. Dan halaman

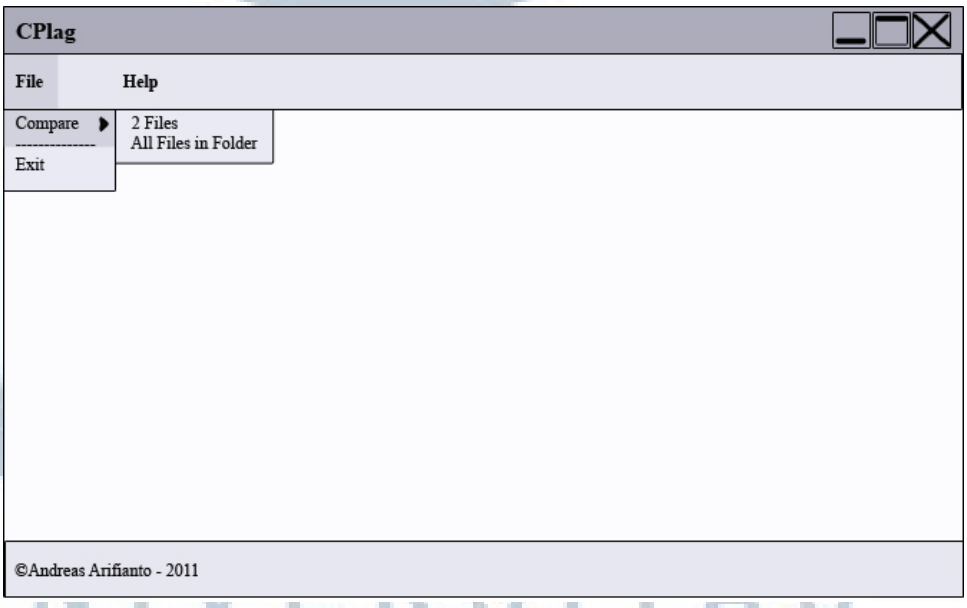
"*About*" berisi informasi secara umum tentang aplikasi yang dibuat serta profil pembuat aplikasi serta tahun pembuatan aplikasi.

3.5.5 Desain Antarmuka

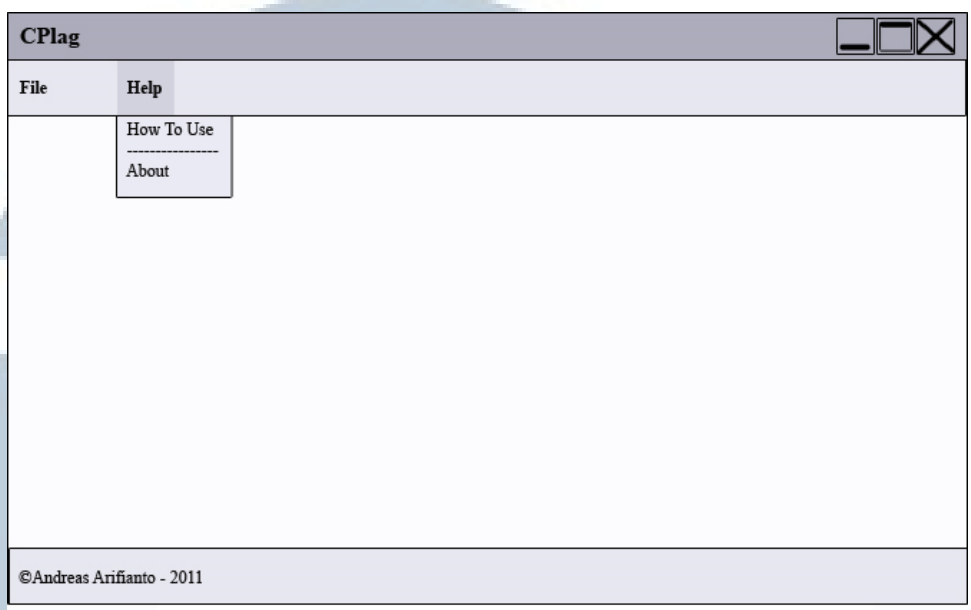
Desain antarmuka (*interface*) yang akan diimplementasikan ke dalam aplikasi yakni.

1. Halaman Utama

   Halaman utama berfungsi sebagai pusat navigasi menu, dimana terdapat menu utama "*File*", yang terdiri dari submenu "*Compare*" dan "*Exit*" serta "*Help*", yang terdiri dari submenu "*How To Use*" dan "*About*".



Gambar 3.11 Desain Antarmuka Halaman Utama Menu "*Compare*"

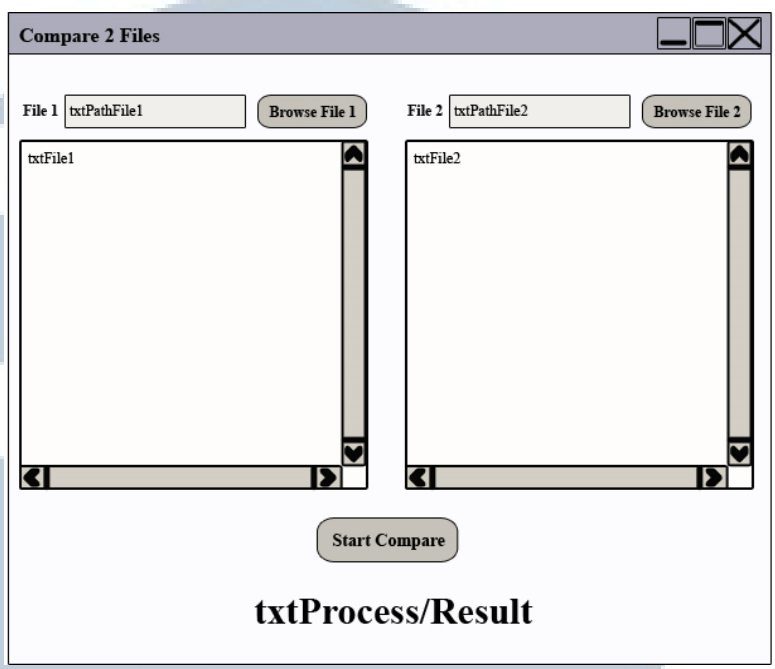


Gambar 3.12 Desain Antarmuka Halaman Utama Menu "*Help*"

2. Halaman "*Compare 2 Files*"

Pada halaman "*Compare 2 Files*" terdapat tombol "*Browse File 1*" dan "*Browse File 2*" yang berfungsi untuk memilih dokumen kode program yang akan dibandingkan. Sedangkan *text box* "*txtPathFile1*" dan "*txtPathFile2*" berfungsi untuk menampung *path* dari dokumen yang dipilih *user*.

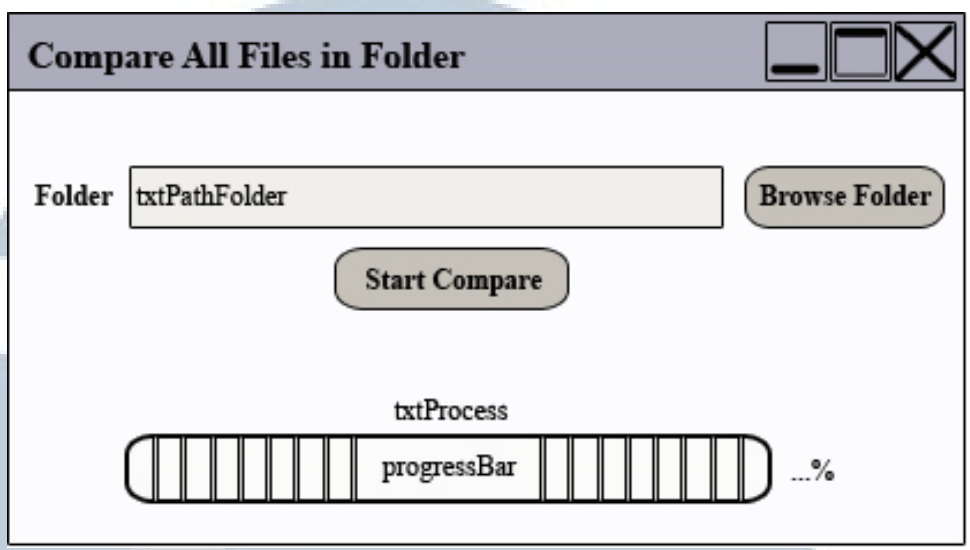
Selain itu, juga terdapat *text box* "*txtFile1*" dan "*txtFile2*" yang berfungsi untuk menampilkan isi dari dokumen kode program yang dipilih. Lalu terdapat pula tombol "*Start Compare*" yang berfungsi untuk memulai proses pendeteksian plagiarisme serta teks "*txtProcess*/*Result*" yang berfungsi untuk menampilkan tahapan proses yang sedang berjalan dan hasil bobot persentase kecenderungan plagiarisme yang dihasilkan.



Gambar 3.13 Desain Antarmuka Halaman "*Compare 2 Files*"

3. Halaman "*Compare All Files in Folder*"

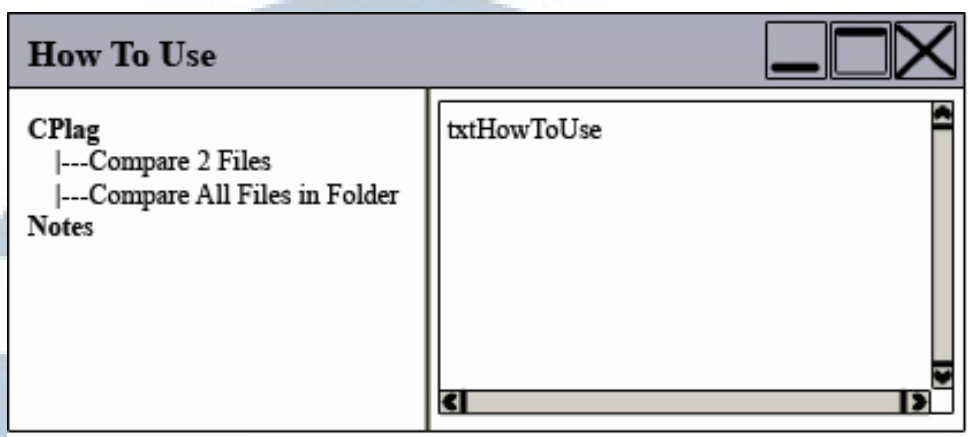
   Pada halaman ini, terdapat tombol "*Browse Folder*" yang berfungsi untuk memilih *folder* yang akan diproses. Sedangkan kolom "*txtPathFolder*" berfungsi untuk menampung *path* dari *folder* yang dipilih *user*. Selain itu, terdapat pula tombol "*Start Compare*" yang berfungsi untuk memulai proses pendeteksian plagiarisme serta teks "*txtProcess*" dan *progress bar* "*progressBar*" yang berfungsi untuk menunjukkan tahapan dari proses yang sedang berjalan.



Gambar 3.14 Desain Antarmuka Halaman "*Compare All Files in Folder*"

4. Halaman "*How To Use*"

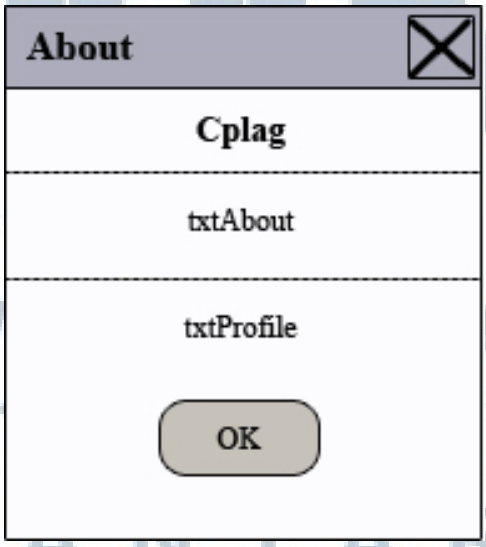
   Halaman "*How To Use*" dibagi atas 2 bagian. Di bagian kiri, terdapat struktur navigasi *tree* yang berfungsi untuk menampilkan pilihan menu yang dapat dipilih *user*, yang terdiri dari 2 pilihan menu utama, yakni Cplag dan *Notes*. Menu *Cplag* sendiri dibagi menjadi 2, sesuai dengan fitur yang terdapat di apliaksi ini, yakni *Compare 2 Files* dan *Compare All Files in Folder*. Sedangkan di bagian kanan, terdapat *text box* "*txtHowToUse*" yang berfungsi untuk menampilkan informasi sesuai dengan menu yang dipilih pada *tree*.



Gambar 3.15 Desain Antarmuka Halaman "*How To Use*"

5. Halaman "*About*"

   Pada halaman ini, terdapat teks "*txtAbout*" yang berisi penjelasan secara umum tentang aplikasi CPlag, yang meliputi teknik dan algoritma yang digunakan di dalam aplikasi. Sedangkan teks "*txtProfile*" berisi informasi mengenai profil pembuat aplikasi, beserta tahun pembuatan aplikasi. Juga terdapat sebuah tombol "OK" yang berfungsi untuk menutup halaman ini.



Gambar 3.16 Desain Antarmuka Halaman "*About*"