# **DESKRIPSI PROGRAM**

Definisi/ maksud program, tujuan, definisi similarity pada program…, spesifikasi program (berapa file ?, bahasa ?, …)

## **A. Data Eksternal ???**

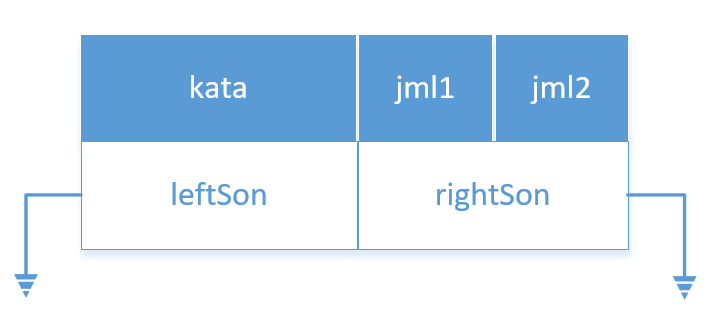
Pertama disiapkan terlebih dahulu minimal 2 file yang akan diperiksa berisi teks, beserta 2 file tambahan. File tambahan pertama adalah kamus kata dasar dan file tambahan kedua adalah kamus *stopword*. *Root word* dan *stop word* disimpan dalam file eksternal agar pengguna tidak perlu mengedit kembali *source code* program.

## **B. Program ???**

Program akan membaca file tambahan terlebih dahulu, file *rootword* dan *stopword*. Masing – masing kamus (file) dibuat menjadi *binary search tree*. Dilanjutkan dengan membaca file pertama yang akan dibaca per kata (*tokenizing*), sembari dibaca dilakukan tahap *case folding*. *Case folding* adalah mengubah huruf kapital menjadi huruf kecil dan menghapus karakter selain huruf.

Kata – kata yang telah dibaca kemudian diperiksa ke dalam tree kamus pertama (kamus *stop word*). Setelahnya diperiksa kata dasarnya pada tree kamus kedua (kamus *root word*). Setelah menjadi kata dasar, barulah kata tersebut dapat dimasukkan ke dalam binary tree file.

Jika kata yang akan dimasukan sudah ada, maka jumlah akan bertambah, jika kata yang sedang dibaca berasal dari file pertama, maka jumlah yang akan bertambah satu adalah jml1, begitupun dengan file kedua. Jika kata yang akan dimasukan belum ada pada *tree*, maka akan dibuat sebagai node baru. *Node* pada *tree* untuk file digambarkan pada gambar 1 berikut.



Gambar 1. Node pada tree untuk file

Langkah di atas, mulai dari *tokenizing, case folding*, sampai memasukan kata ke *tree* file, dilakukan berulang sampai akhir file pertama lalu ke file kedua. Dengan selesainya membaca file dan membangun tree, maka selesailah tahap *pre processing*.

Dilanjutkan dengan tahap perhitungan, yakni dengan melakukan kunjungan (*traverse*) pada setiap *node* di *tree* file yang baru saja dibangun. *Traversal* dilakukan untuk mencari nilai statistik terkecil dan terbesar dari satu kata, antara file pertama dan file kedua. Nilai yang terkecil dari *node* akan dijumlahkan bersama nilai dari *node* lain yang lebih kecil juga, begitupun dengan nilai yang nilai yang terbesar dari *node*.

Setelah disimpan nilai terbesar dan terkecil dari semuanya, maka presentase kemiripan kedua file adalah perbandingan nilai terbesar dan terkecil. Atau dapat dilihat pada rumus berikut.

**RANCANGAN PROGRAM**

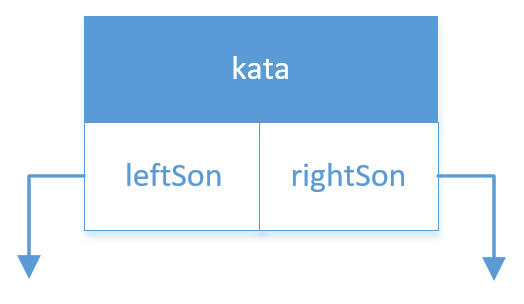
## **Rancangan Struktur Data**

Program ini, memiliki kebutuhan data untuk menyimpan informasi yang akan diproses. Penyimpanan data ini terbagi menjadi dua, eksternal dan internal. Penyimpanan eksternal dilakukan untuk dua kepentingan : file yang akan diperiksa dan kamus atau database (*stopword* dan *rootword*). Sementara penyimpanan internal digunakan selama program berlangsung untuk manipulasi data.

Adapun penyimpanan internal digunakan untuk menampung data dari file eksternal. Penggunaan penyimpanan memori internal terbagi menjadi beberapa bagian. Bagian – bagian tersebut dijelaskan sebagai berikut :

### **1. Penyimpanan Kamus (*Database*)**

Kamus yang berisi kumpulan kata – kata (baik untuk *stopword* maupun *rootword*), akan dibaca oleh program kemudian dibangun menjadi sebuah AVL *Tree*. Pemilihan struktur AVL *Tree* bukan tanpa alasan, hal ini dilakukan agar ketika melakukan pencarian kata ke kamus (*database*) waktu eksekusinya lebih kecil dibanding menggunakan struktur lain. Secara teori, pencarian pada tree paling buruk akan memakan waktu sebesar O ( ).

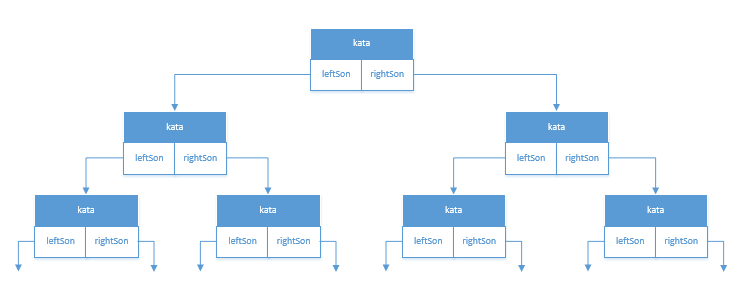


Gambar 2. Simpul pada tree untuk kamus

Gambar 2 menggambarkan struktur simpul pada *tree* untuk kamus (*database*). Simpul hanya terdiri dari variabel kata yang merupakan *String*, lalu leftSon dan rightSon yang merupakan *pointer* bertipe simpul ini.

Tabel 1 Variabel pada simpul

|  |  |  |
| --- | --- | --- |
| Tipe Data | Nama Variabel | Keterangan |
| string | kata | menyimpan kata yang sudah melalui *preprocessing* |
| pointer | leftSon | menunjuk *node* anak kiri |
| pointer | rightSon | menunjuk *node* anak kanan |



Gambar 3. Struktur tree yang dibangun dari simpul - simpul kamus

Tabel 1 menjelaskan variabel – variabel yang terdapat pada node atau simpul kamus. Simpul – simpul ini kemudian disusun menjadi sebuah tree seperti pada gambar 3.

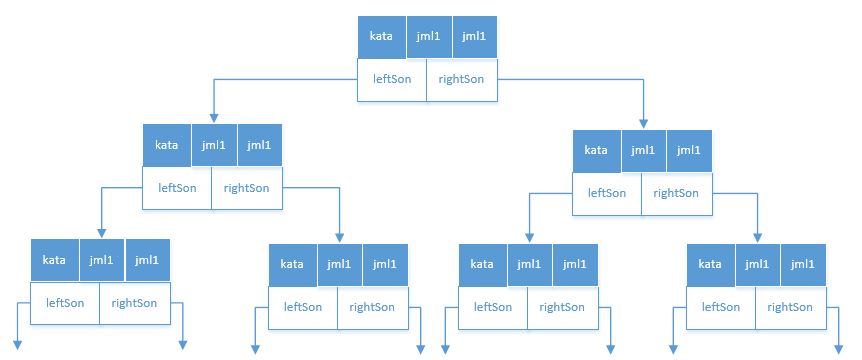
### **2. Penyimpanan File Teks**

File akan dibaca pertama kali oleh program kemudian dibangun menjadi sebuah AVL *Tree*, kurang lebihnya sama seperti struktur data kamus. Perbedaannya terletak pada struktur simpul. Struktur simpul untuk tree file ini dijelaskan pada gambar 1 (BAB III). Variabel yang menyusun simpul ini antara lain :

Tabel 2 Variabel penyusun simpul tree

|  |  |  |
| --- | --- | --- |
| Tipe Data | Nama Variabel | Keterangan |
| string | kata | menyimpan kata yang sudah melalui *preprocessing* |
| int | jml1 | menyimpan jumlah kata yang sama di file pertama |
| int | jml2 | menyimpan jumlah kata yang sama di file kedua |
| pointer | leftSon | menunjuk *node* anak kiri |
| pointer | rightSon | menunjuk *node* anak kanan |

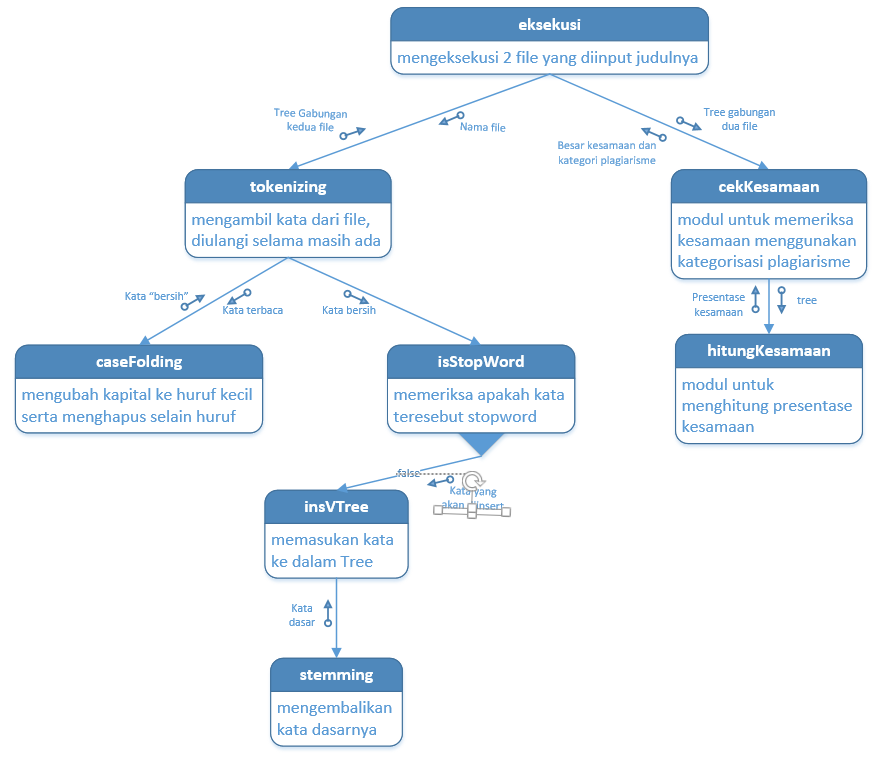
Simpul – simpul ini kemudian dibangun menjadi sebuah AVL *Tree*. Gambaran *tree* dapat dilihat pada gambar 4 berikut.



Gambar 4. AVL Tree untuk file

## **4.2. Rancangan Struktur Proses**

Pada subbab ini akan dijelaskan modul – modul yang digunakan beserta alur kerja program. Struktur proses dapat dilihat pada gambar 5, dan dijabarkan pada tabel – tabel dibawahnya.



Gambar 5 Struktur Proses

|  |  |  |
| --- | --- | --- |
| Nama Modul | | tokenizing |
| Jenis | | procedure |
| Author | |  |
| Deskripsi | | Membaca file sampai habis dan mengambil setiap katanya. Memanfaatkan keberadaan karakter spasi sebagai pemisah antar kata |
| Initial State | | *Tree* file belum terbentuk |
| Final State | | *Tree* file terbentuk |
| Parameter | Input | *Tree* Kamus |
| Output | - |
| Input/Output | File yang akan dibaca  *Tree* file |
| Kamus Data (Variabel Lokal) | | |
|  |  |  |
| Algoritma | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| Nama Modul | | caseFolding |
| Jenis | | function |
| Author | |  |
| Deskripsi | | Mengembalikan string menjadi kata yang “bersih”, dalam artian semua hurufnya tidak kapital dan tidak ada karakter selain huruf |
| Initial State | | String input belum diketahui “bersih” atau tidaknya |
| Final State | | Mengembalikan string yang “bersih” |
| Parameter | Input | *String* kata |
| Output | - |
| Input/Output | - |
| Kamus Data (Variabel Lokal) | | |
|  |  |  |
| Algoritma | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| Nama Modul | | isStopWord |
| Jenis | | function |
| Author | |  |
| Deskripsi | | Memeriksa apakah kata yang diinputkan merupakan *stopword* atau bukan. |
| Initial State | | - |
| Final State | | Mengembalikan nilai true / false |
| Parameter | Input | *Tree* Kamus String kata yang akan diperiksa |
| Output | - |
| Input/Output | - |
| Kamus Data (Variabel Lokal) | | |
|  |  |  |
| Algoritma | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| Nama Modul | | Stemming |
| Jenis | | function |
| Author | |  |
| Deskripsi | | Mengembalikan kata dasar dari kata yang diinputkan. |
| Initial State | | sembarang |
| Final State | | Mengembalikan kata dasar |
| Parameter | Input | *Tree* Kamus  String kata yang akan diubah ke kata dasar |
| Output | - |
| Input/Output | - |
| Kamus Data (Variabel Lokal) | | |
|  |  |  |
| Algoritma | | |
|  | | |

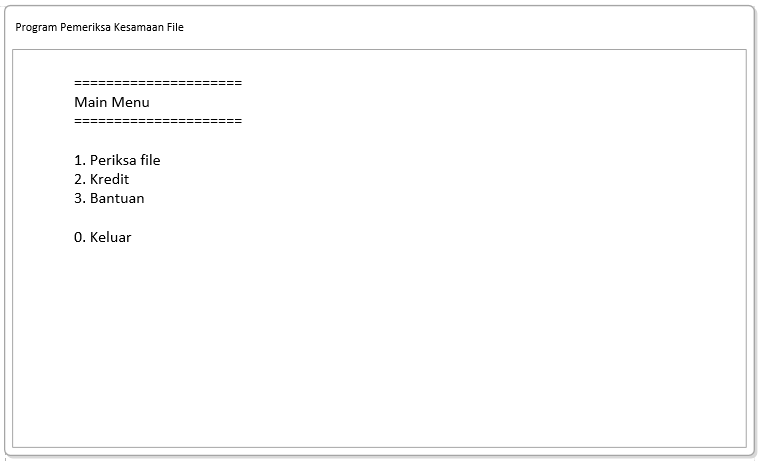
|  |  |  |
| --- | --- | --- |
| Nama Modul | | hitungStatistik |
| Jenis | | procedur |
| Author | |  |
| Deskripsi | | Mencari jumlah kata yang terbanyak dan terkecil yang muncul, jumlah terbanyak dan jumlah terkecil ini kemudian akan digunakan untuk perhitungan kesamaan. |
| Initial State | | sembarang |
| Final State | | Terhitung jumlah yang terbanyak dan yang terkecil |
| Parameter | Input | - |
| Output | - |
| Input/Output | integer max  integer min |
| Kamus Data (Variabel Lokal) | | |
|  |  |  |
| Algoritma | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| Nama Modul | | hitungMin |
| Jenis | | function |
| Author | |  |
| Deskripsi | | Merupakan algoritma rekursif untuk mengembalikan nilai minimum antara 2 bagian jumlah pada node tree file. |
| Initial State | | sembarang |
| Final State | | Mengembalikan nilai minimum antara jml1 dan jml2 |
| Parameter | Input | node yang akan diperiksa |
| Output | - |
| Input/Output | - |
| Kamus Data (Variabel Lokal) | | |
|  |  |  |
| Algoritma | | |
|  | | |

|  |  |  |
| --- | --- | --- |
| Nama Modul | | hitungMax |
| Jenis | | function |
| Author | |  |
| Deskripsi | | Merupakan algoritma rekursif untuk mengembalikan nilai maximum antara 2 bagian jumlah pada node tree file. |
| Initial State | | sembarang |
| Final State | | Mengembalikan nilai maximum antara jml1 dan jml2 |
| Parameter | Input | node yang akan diperiksa |
| Output | - |
| Input/Output | - |
| Kamus Data (Variabel Lokal) | | |
|  |  |  |
| Algoritma | | |
|  | | |

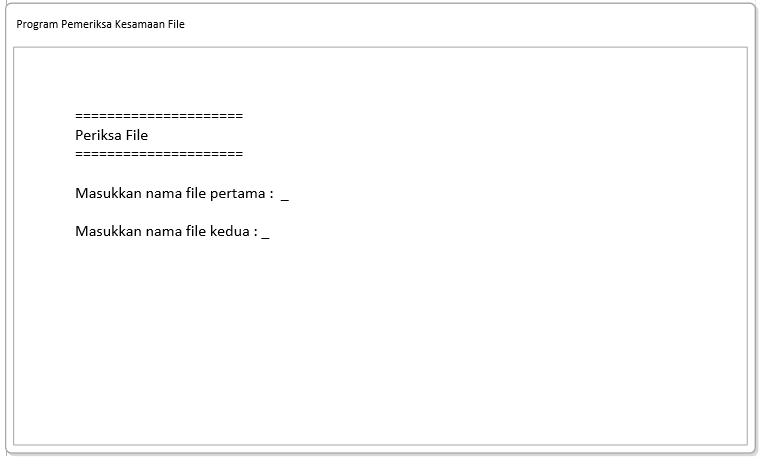
## **4.3. Rancangan Tampilan**

Pada subbab ini akan dijelaskan *mockup* atau desain tampilan program. Pertama kali program dijalankan, maka akan masuk ke tampilan main menu seperti pada gambar 6 berikut.



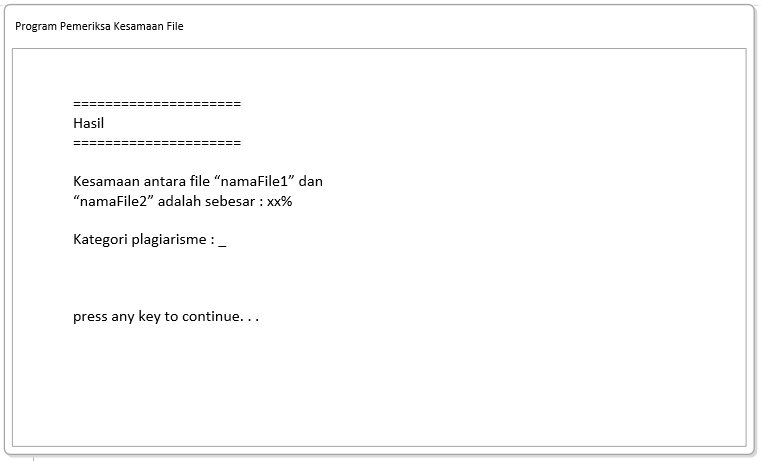
Gambar 6. Tampilan Main Menu

Ketika user memilih 1, maka akan dilanjutkan ke tampilan berikutnya seperti pada gambar 7. Jika user memilih 2, maka akan ditampilkan kredit pembuat aplikasi ini, sementara jika user memilih 3, maka akan ditampilkan bantuan atau *user guide* untuk menggunakan aplikasi ini. Jika user memilih 0 maka program akan berhenti.



Gambar 7. Tampilan program

Pada saat user ingin memeriksa kesamaan file, maka program akan meminta input berupa nama file pertama dan kedua yang akan dibandingkan. Selanjutnya, akan dijalankan proses perhitungan dan barulah muncul tampilan seperti pada gambar 8 berikut.



Gambar 8. Tampilan Hasil

Pada tampilan akhir ini, dimunculkan hasil perbandingan kedua file. Hasil perbandingan berupa presentase kemiripan antara dua file tersebut. Sementara kategori plagiarisme didapat menggunakan pengecekan kondisional berdasarkan presentase kemiripan yang dicocokan dengan aturan kategorisasi plagiarisme.

# **TEKNIS KEGIATAN**

## **A. Pembagian Tugas**

Langkah pertama sebelum mengerjakan program adalah membagi tugas antar anggota kelompok. Tugas tiap anggota tim dijabarkan dalam tabel 3.

Tabel 3 Pembagian Tugas Kelompok

|  |  |  |
| --- | --- | --- |
| Nama | Tugas | Deadline |
| Fadly | Membuat ADT dan Algoritma AVL Tree | 20 Juni 2018 |
| Afif | Membuat algoritma tokenizing, caseFolding, dan stopword | 20 Juni 2018 |
| Rizki | Membuat algortima stemming | 20 Juni 2018 |

## **B.** **Jadwal Kegiatan**

Adapun jadwal pengerjaan program dapat dilihat pada tabel 4 berikut.

Tabel 4 Jadwal Kegiatan

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| No. | Jenis Kegiatan | Pekan ke- | | | | |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Membuat rancangan data dan modul yang akan digunakan |  |  |  |  |  |
| 2 | Mencari dan eksplorasi algoritma *stemming* |  |  |  |  |  |
| 3 | Mengerjakan program (+laporan perkembangan tiap pekan) |  |  |  |  |  |
| 4 | Uji coba awal (mengukur berjalan atau tidaknya program) |  |  |  |  |  |
| 5 | Membuat laporan |  |  |  |  |  |
| 6 | Masa perbaikan |  |  |  |  |  |
| 7 | Uji coba (mengukur efisiensi) |  |  |  |  |  |
| 8 | Finalisasi |  |  |  |  |  |
| 9 | Demo Program |  |  |  |  |  |
| 10 | *Perbaikan (masukan dari Dosen)* |  |  |  |  |  |