BAB III

ANALISIS DAN PERANCANGAN APLIKASI

3.1 Analisis Permasalahan

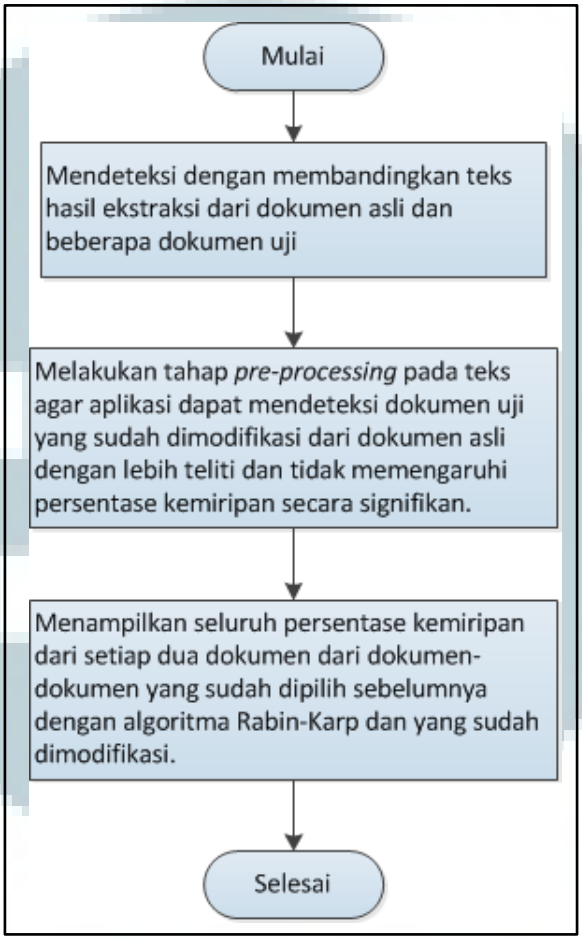
Untuk mencegah tindakan plagiarisme, khususnya dalam dunia akademik, pendeteksian tindakan plagiarisme terhadap dokumen teks yang dilakukan oleh tenaga pengajar merupakan salah satu solusi yang baik. Namun, proses pendeteksian yang dilakukan secara manual akan sangat sulit dan menghabiskan waktu yang banyak karena jumlah teks yang tidak sedikit dan teks yang diplagiat tidak selalu sama dengan aslinya. Orang yang melaksanakan proses pendeteksian, khususnya tenaga pengajar, tidak selalu memiliki waktu luang untuk melakukannya.

Permasalahan di atas dapat diselesaikan dengan membuat sebuah aplikasi yang dapat mendeteksi plagiarisme pada dokumen teks. Namun, itu bukan hal yang mudah karena ada kemungkinan bahwa teks dari dokumen teks yang akan dibandingkan tidak selalu sama dan dapat bervariasi sehingga dapat memengaruhi hasil pendeteksian. Oleh karena itu, aplikasi harus dirancang dengan mengimplementasikan metode-metode yang tepat agar dapat mendeteksi plagiarisme secara akurat dan cepat tanpa terpengaruh oleh hal tersebut.

3.2 Usulan Solusi Permasalahan

Solusi yang dapat diusulkan adalah membuat aplikasi pendeteksi plagiarisme menggunakan algoritma Rabin-Karp sehingga pendeteksian dapat

dilakukan dengan efektif dan efisien. Berikut ini adalah *flowchart* dari hal-hal yang bisa dilakukan oleh aplikasi.



Gambar 3.1 *Flowchart* Umum Aplikasi

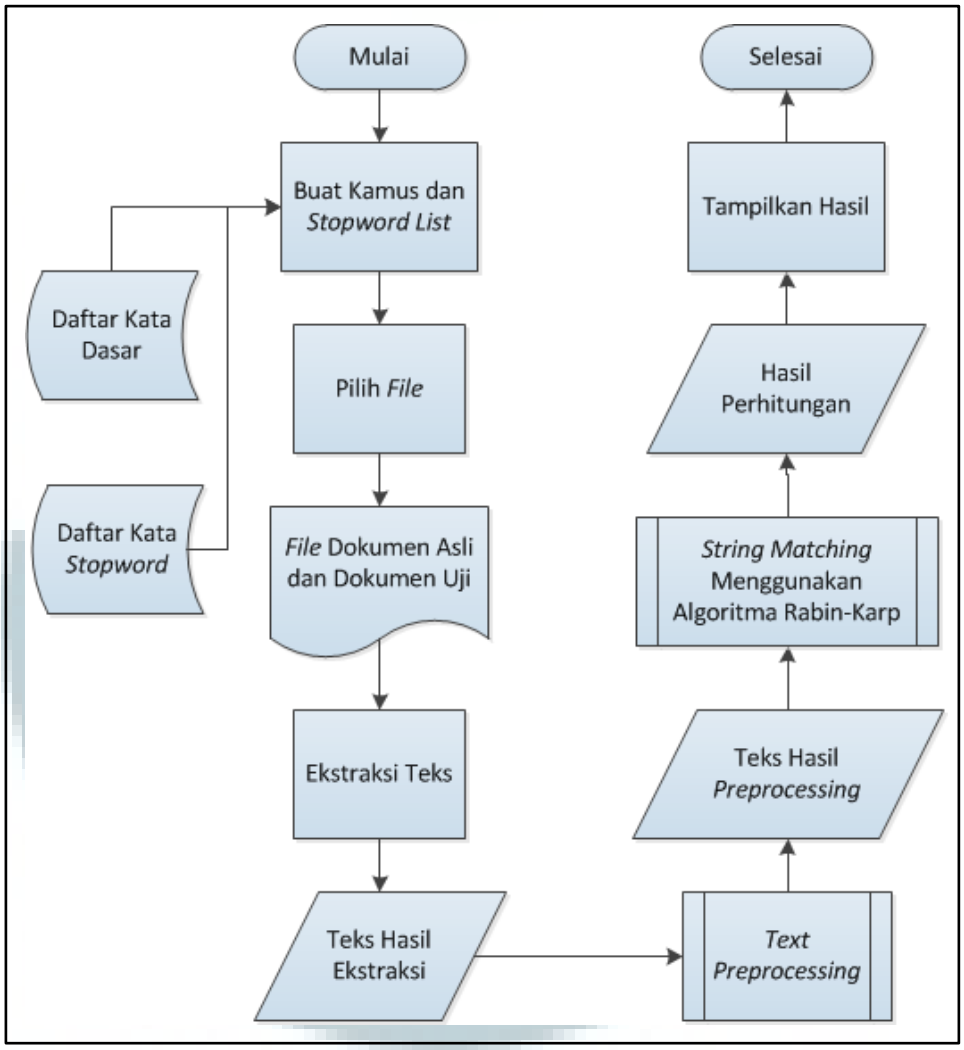
Algoritma Rabin-Karp merupakan algoritma pencocokan *string* (*string matching*) yang diciptakan oleh Michael O. Rabin dan Richard M. Karp. Prinsip kerja dari algoritma ini adalah mencari pola (*pattern*) *string* dengan membandingkan nilai *hash* dari setiap *string*. Pencocokan banyak pola menjadi kelebihan dari algoritma ini(Gupta, Agarwal, & Varshney, 2010). Algoritma ini juga memiliki sedikit modifikasi, yaitu perubahan pada syarat dalam mencocokkan *string*.

Untuk dapat mengimplementasikan pencocokan banyak pola, aplikasi akan memakai konsep *n-grams.* Sebelum membentuk *n-grams,* hal yang perlu diperhatikan adalah teks yang dibandingkan dapat berbeda. Itulah sebabnya aplikasi ini membutuhkan tahap *pre-processing* sehingga kedua teks memiliki struktur teks yang sama. Dalam *pre-processing* ini dibutuhkan kamus untuk *stemming* dan *stop word list* untuk *filtering*. *N-grams* merupakan kata-kata dibentuk dengan mengambil potongan-potongan kata dengan panjang tertentu secara kontinu dan tumpang tindih dari awal sampai akhir dari teks (Náther, 2005). Dengan begitu, *n-grams* bisa digunakan sebagai pola *string* yang banyak sehingga dapat digunakan dalam pencocokan banyak pola. Setiap pola akan diubah menjadi nilai *hash* dan dibandingkan nilainya. Setelah pencocokan selesai, akan diketahui pola mana saja yang sama dari kedua teks dan kemudian hasil persentase kemiripan dihitung dengan rumus *Dice's coefficient*.

Aplikasi juga memiliki modul penambahan kata baru ke dalam kamus atau *stop word list.* Kedua daftar kata ini dipakai dalam tahap *pre-processing.* Modul ini tidak ada di dalam gambar 3.1 karena modul ini tidak memiliki dependensi dengan modul pendeteksi plagiarisme. Artinya, bisa langsung menambahkan kata baru tanpa harus mendeteksi terlebih dahulu.

3.3 Spesifikasi Umum Aplikasi

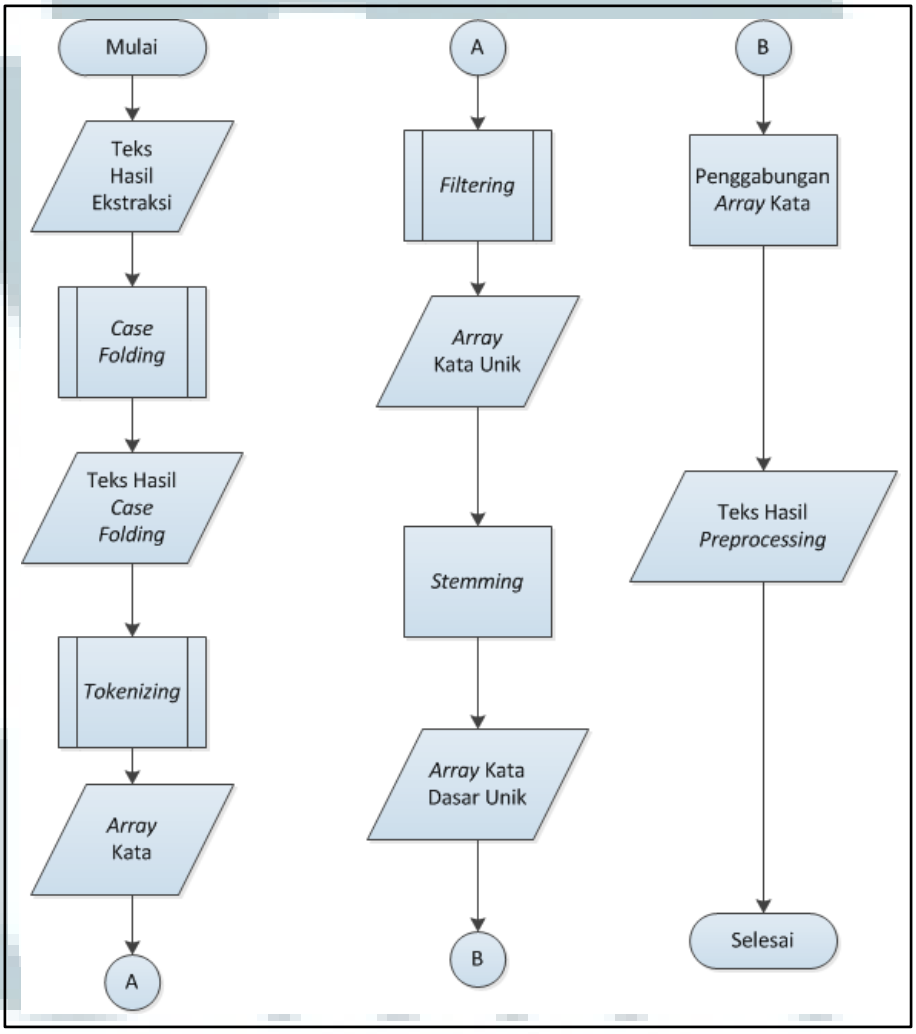
Tujuan dari pembuatan aplikasi ini adalah untuk mendeteksi plagiarisme pada dokumen teks berbahasa Indonesia dengan algoritma Rabin-Karp. Langkah-langkah untuk mendeteksi plagiarisme dalam aplikasi ini lebih mudah dijelaskan dengan *flowchart* berikut.



Gambar 3.2 *Flowchart* Mendeteksi Plagiarisme

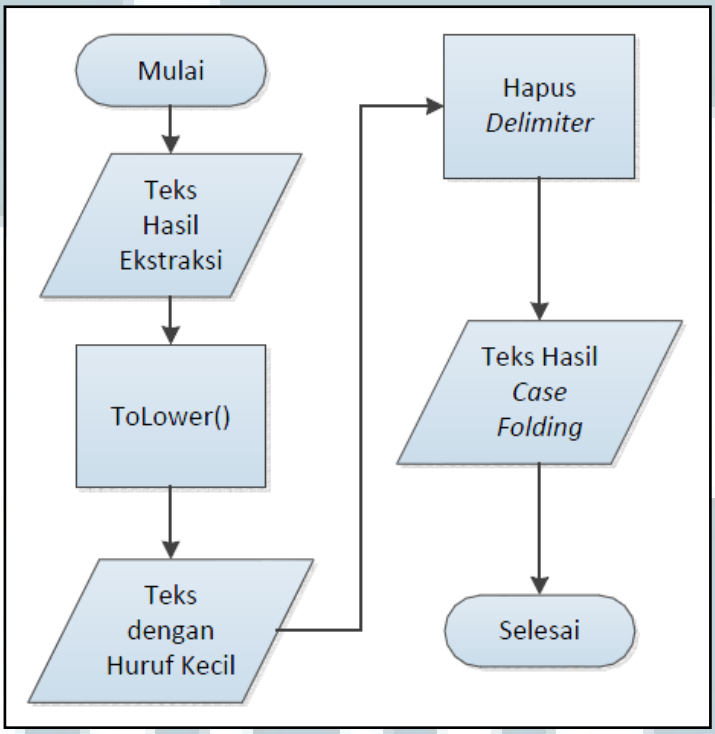
Ketika aplikasi dibuka, akan dibuat kamus yang berisi kata dasar dan *stop word list* dalam bahasa Indonesia. Daftar kata diambil dari dua buah *text file* eksternal yang masing-masing berisi kata dasar dan *stop word*. Keduanya akan digunakan dalam tahap *pre-processing*. Kemudian, *user* akan diminta untuk membuka *file* dokumen asli dan dokumen uji. Dokumen asli yang dipilih hanya bisa satu, sedangkan dokumen uji yang dipilih bisa lebih dari satu. Ekstensi *file* yang bisa dipilih adalah .txt, .doc, .docx, dan .pdf. Pengguna tidak harus memilih format *file* yang sama. Misalkan ingin membandingkan *file* .doc dengan .pdf. Teks

akan diekstrak dari seluruh dokumen. Hasil ekstraksi akan memasuki tahap *pre-processing* yang sudah meliputi *case folding, tokenizing, filtering*, dan *stemming*. Setelah itu, hasil dari tahap *pre-processing* akan dipakai dalam pencocokan *string* oleh algoritma Rabin-Karp. Pembentukan *n-grams*, *hashing*, dan perhitungan *similarity* sudah termasuk dalam proses pencocokan. *Output* aplikasi berupa persentase kemiripan dan waktu proses dan bisa disimpan ke dalam suatu *text file.*



Gambar 3.3 *Flowchart* Tahap *Pre-processing*

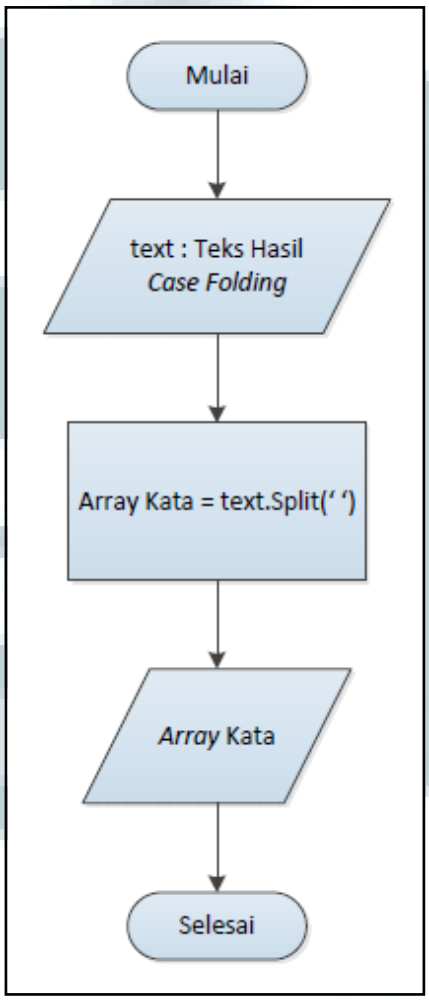
Hasil ekstraksi berupa teks akan diolah dalam tahap *pre-processing* yang meliputi empat proses, yaitu *case folding*, *tokenizing*, *filtering*, dan *stemming*. *Case folding* akan menghasilkan teks berhuruf kecil tanpa karakter selain huruf dan spasi. *Tokenizing* akan membagi teks menjadi kata-kata yang disimpan dalam sebuah *array*. Setelah itu, *array* kata tersebut akan melewati proses *filtering* yang mana akan menghilangkan kata yang kurang berarti. Kemudian terdapat *stemming* yang akan mengubah kata berimbuhan menjadi kata dasar. *Array* kata tersebut akan digabungkan menjadi satu teks dan akan diproses selanjutnya. *Flowchart* tahap *pre-processing* digambarkan pada gambar 3.3.



Gambar 3.4 *Flowchart* Proses *Case Folding*

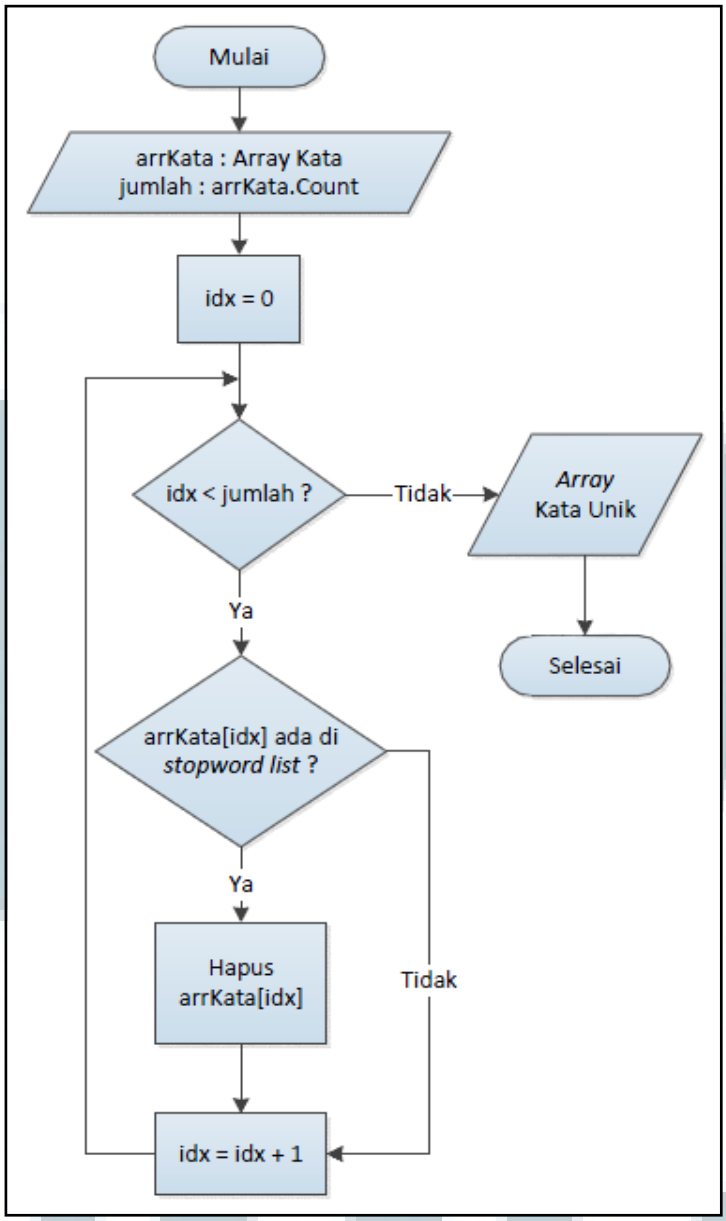
Proses pertama pada tahap *pre-processing* adalah *case folding* yang bertujuan untuk mengubah semua huruf menjadi huruf kecil.Huruf dalam teks hasil ekstraksi yang diubah adalah mulai dari huruf ‘a’ sampai dengan ‘z’. Selain

itu, dianggap *delimiter* dan akan dihilangkan kecuali spasi. *Flowchart* dari proses *case folding* digambarkan pada gambar 3.4.



Gambar 3.5 *Flowchart* Proses *Tokenizing*

Gambar 3.5 adalah *flowchart* proses *tokenizing*. Proses ini menerima *input* berupa teks hasil *case folding* akan dibagi berdasarkan spasi. Setelah itu, *output* yang dihasilkan adalah *array* yang berisi kata dari teks.

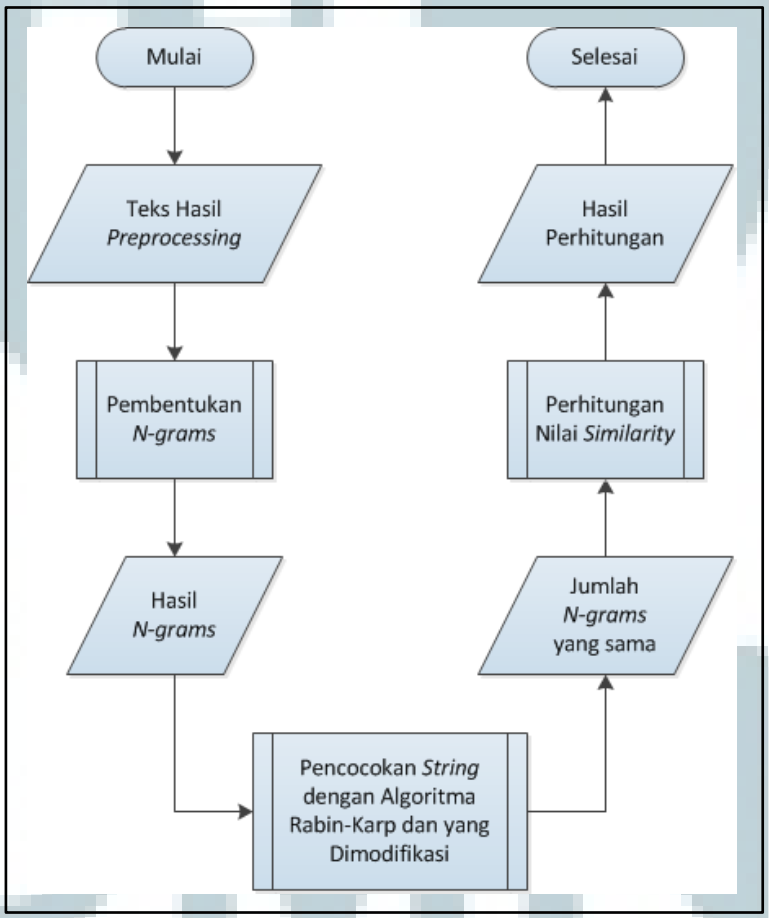


Gambar 3.6 *Flowchart* Proses *Filtering*

Proses selanjutnya adalah *filtering*. Proses ini akan menghapus kata-kata yang dianggap tidak penting seperti “yang”, “di”, dan “dan” sehingga nantinya hanya kata-kata yang memiliki arti penting saja yang akan diproses. Aplikasi ini menggunakan algoritma *stop word*dimana setiap kata akan dicek apakah kata

tersebut ada dalam *stop word list*. Jika ada, kata tersebut akan dihilangkan sehingga setelah dilakukan proses *filtering* akan didapatkan daftar kata unik.

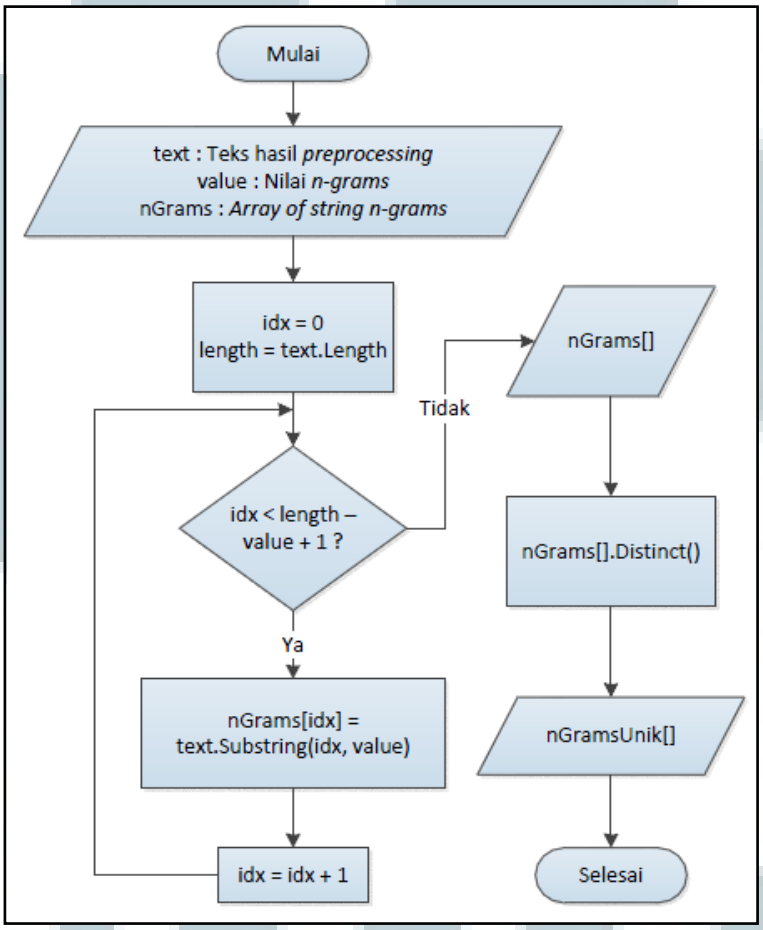
Proses terakhir dalam tahap *pre-processing* adalah *stemming*. Dalam proses ini, seluruh kata yang memiliki *prefix* dan *suffix* akan dihapus sehingga didapatkan kata dasar. Dalam tahap ini, kamus yang telah dibuat sebelumnya akan dipakai. Setelah seluruh proses dalam tahap *pre-processing* selesai, *array* kata akan digabung menjadi satu teks untuk diolah pada tahap berikutnya.



Gambar 3.7 *Flowchart String Matching* Menggunakan Algoritma Rabin-Karp

Untuk mengetahui apakah suatu dokumen plagiat atau tidak, aplikasi akan melakukan tiga proses.Proses pertama adalah pembentukan *n-grams* dari teks

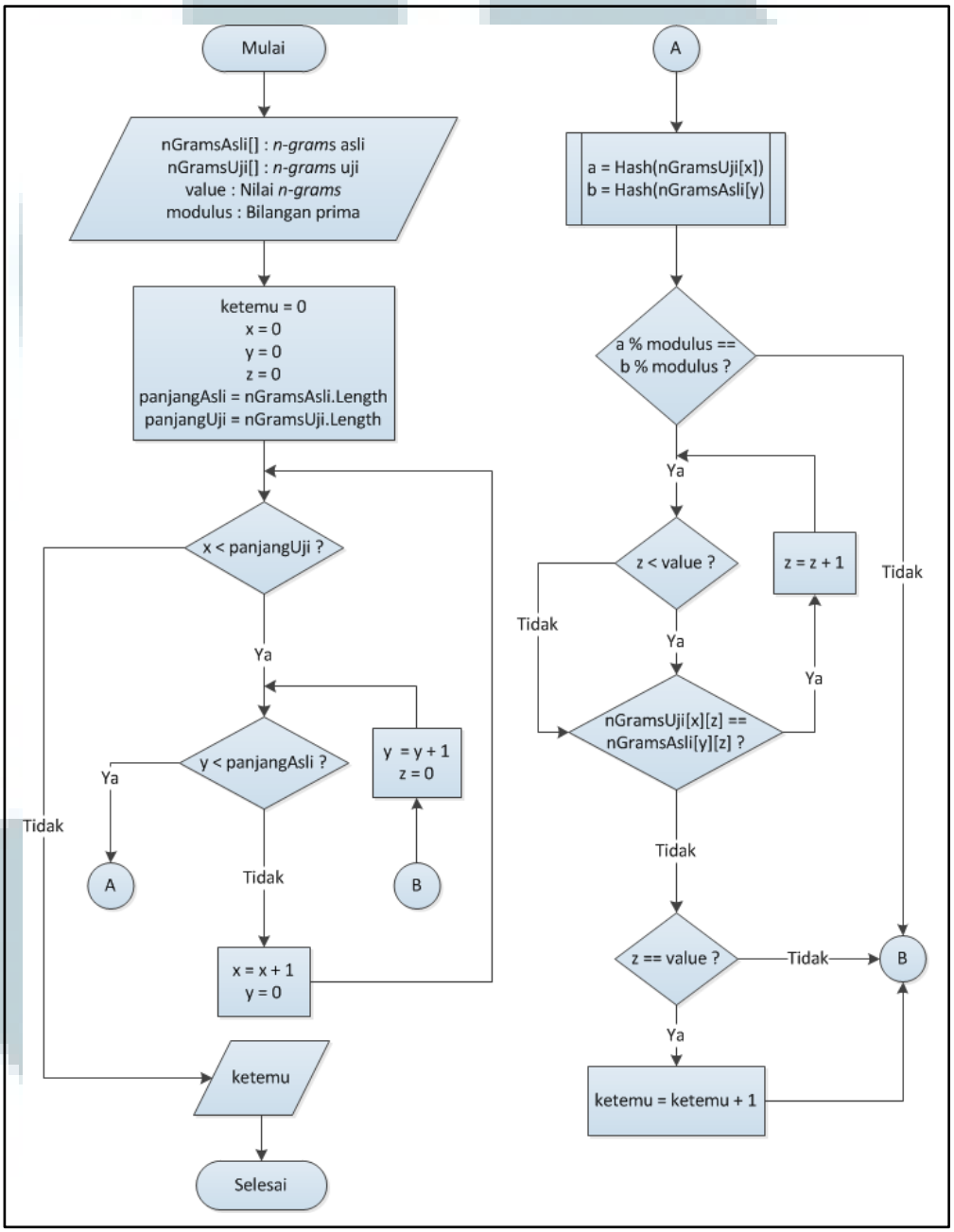
hasil *pre-processing*. Hasil *n-grams* akan dipakai dalam proses selanjutnya yaitu pencocokan *string* dengan algoritma Rabin-Karp dan yang dimodifikasi. Dengan pencocokan *string* ini, jumlah *n-grams* yang sama didapatkan dan akan dilakukan perhitungan nilai *similarity*.



Gambar 3.8 *Flowchart* Proses Pembentukan *N-grams*

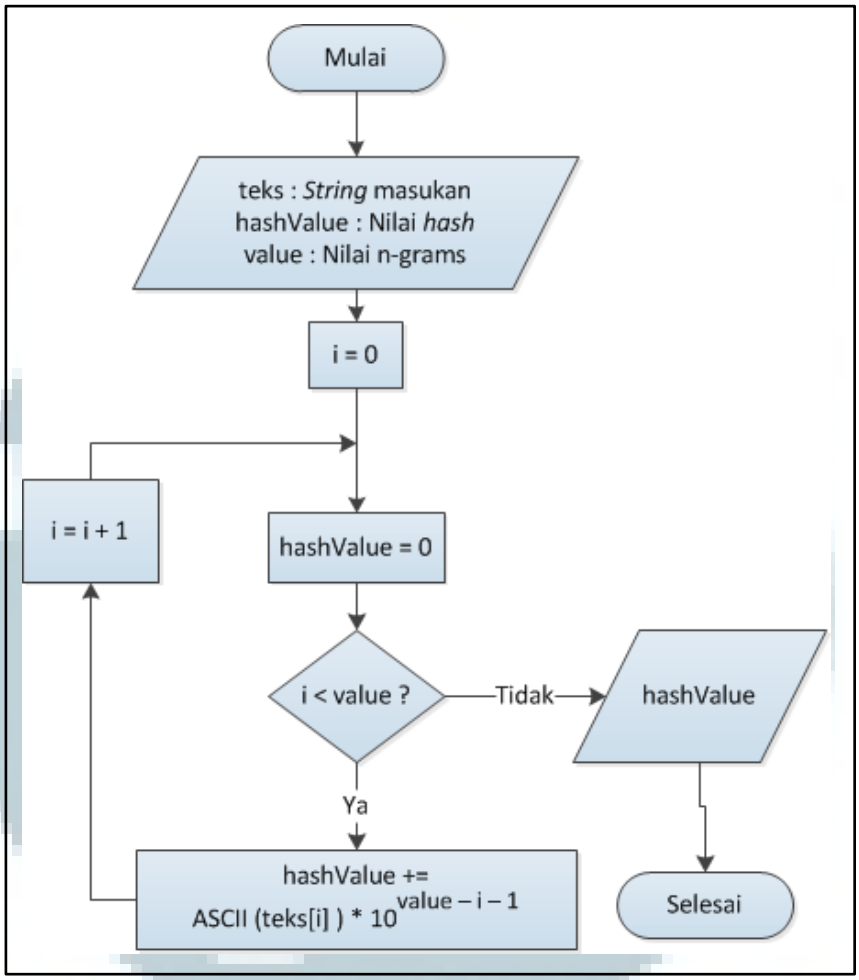
Teks yang sudah melewati tahap *pre-processing* akan diolah lagi agar bisa diproses dalam algoritma Rabin-Karp, yaitu pembentukan *n-grams*. *N-grams* dibentuk dengan memotong kata dengan panjang karakter tertentu secara kontinu dari awal sampai akhir teks. Panjang atau nilai dari *n-grams* dapat ditentukan antara satu dan lima. Setelah itu, *n-grams* yang sudah terbentuk akan digunakan

sebagai pola pencarian dalam algoritma. Kata dalam *n-grams* tidak ada yang berulang. Dalam aplikasi, cara yang digunakan untuk pembentukan *n-grams* adalah dengan *overlapping n-grams*. *Flowchart* pembentukan *n-grams* digambarkan pada gambar 3.8.



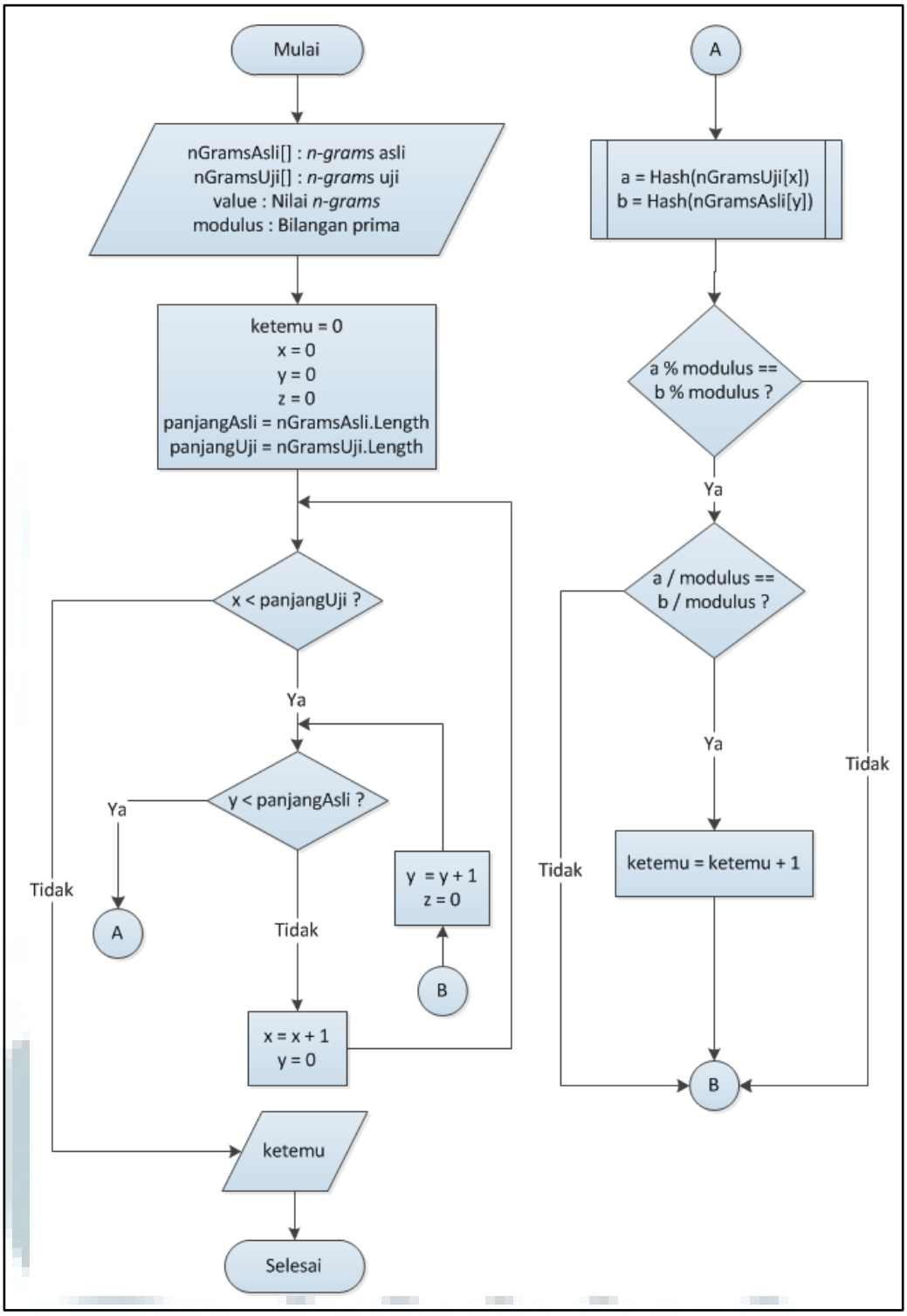
Gambar 3.9 *Flowchart* Algoritma Rabin-Karp

N-grams dihasilkan berupa *array of string*. Setelah seluruh *n-grams* didapatkan, aplikasi akan melakukan pencocokan *multiple pattern* dengan algoritma Rabin-Karp. Waktu proses algoritma ini akan dicatat dan kemudian ditampilkan setelah selesai. Untuk mengimplementasikan algoritma Rabin-Karp, langkah awal yang dilakukan adalah mendefinisikan variabel ketemu dengan nilai awal nol untuk menentukan jumlah *n-grams* yang sama dan bilangan modulus berupa bilangan prima. Langkah berikutnya adalah menentukan nilai ketemu dengan melakukan perbandingan antara masing-masing nilai *hash* dari *n-grams* yang dibentuk dari teks dokumen asli dan nilai *hash* dari *n-grams* yang dibentuk dari teks dokumen uji. Oleh karena itu, dibutuhkan dua iterasi. Iterasi pertama digunakan untuk mengambil seluruh *string* dari *n-grams* yang dibentuk dari teks dokumen uji, sedangkan iterasi kedua digunakan untuk mengambil seluruh *string* dari *n-grams* yang dibentuk dari teks dokumen asli. Dalam iterasi kedua akan dilakukan pengubahan *string* menjadi nilai *hash* dan kemudian dibandingkan. Jika hasil modulus dari kedua nilai *hash* adalah sama, akan dilakukan pengecekan terhadap setiap karakter pada kedua *string*. Jika ada karakter yang berbeda, terjadi *spurious hit* dan nilai ketemu tidak bertambah. Jika seluruh karakter sama, terjadi *successful hit* dan nilai ketemu akan ditambah satu. Jika kedua iterasi selesai, algoritma Rabin-Karp akan mengembalikan nilai ketemu yang nantinya akan digunakan untuk menghitung nilai *similarity*.



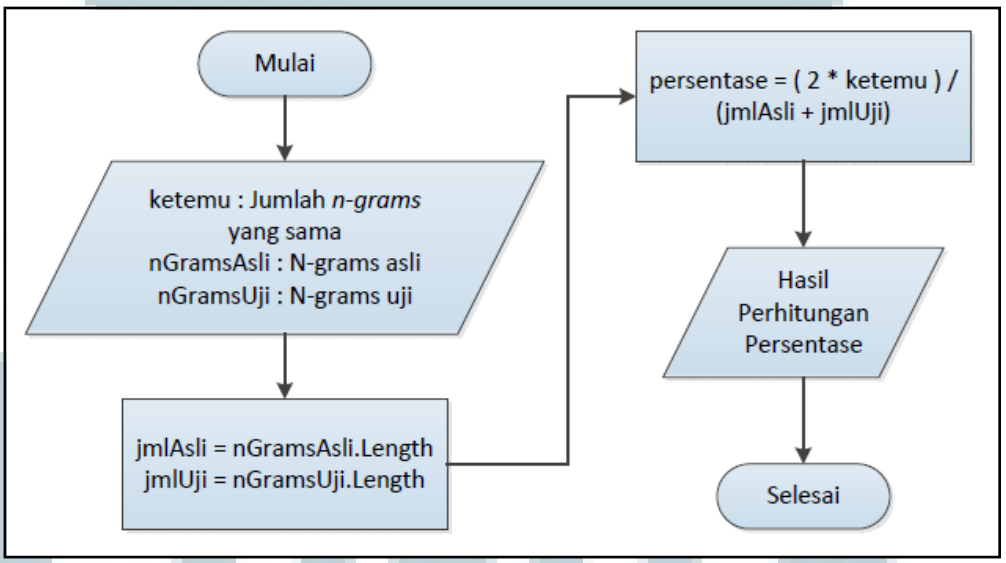
Gambar 3.10 *Flowchart Hashing*

Pada gambar 3.9, setiap *string* dari *n-grams* yang diambil akan diubah menjadi nilai *hash* menggunakan sebuah *hash function.* Fungsi ini dibuat berdasarkan rumus 2.2. Sebuah *string* diambil dari kumpulan *n-gram*. Setelah itu, terdapat sebuah iterasi yang mengambil karakter dari awal sampai akhir *string* dan mengubahnya menjadi bilangan ASCII. Bilangan ini akan dikalikan dengan sepuluh, kemudian dipangkatkan dengan nilai *n-grams* dikurangi iterasi ke berapa dikurangi satu dan disimpan ke dalam sebuah variabel. Nilai variabel ini akan menjadi nilai *hash* dari suatu *string.* Gambar 3.10 merupakan *flowchart* dari *hash function* tersebut.



Gambar 3.11 *Flowchart* Algoritma Rabin-Karp yang Dimodifikasi

Selain menggunakan algoritma Rabin-Karp, aplikasi juga akan menggunakan algoritma Rabin-Karp yang sudah dimodifikasi. Ini artinya terdapat sedikit perbedaan dari algoritma Rabin-Karp sebelumnya. Perbedaan tersebut adalah ketika membandingkan nilai *hash*. Seperti yang ditulis sebelumnya, dalam mencocokkan *string* masih dapat terjadi *spurious hit*. Ini tentu akan memperlambat proses pencocokan *string* karena diperlukan operasi pengecekan karakter jika nilai *hash* yang dibandingkan sama. Salah satu upaya untuk menghilangkan kemungkinan terjadinya *spurious hit* adalah dengan memodifikasi algoritma. Oleh karena itu, algoritma Rabin-Karp ditambahkan satu syarat dalam membandingkan yaitu hasil bagi dengan modulus dari kedua nilai *hash* harus sama. Dengan demikian, tidak lagi diperlukan operasi pengecekan karakter.

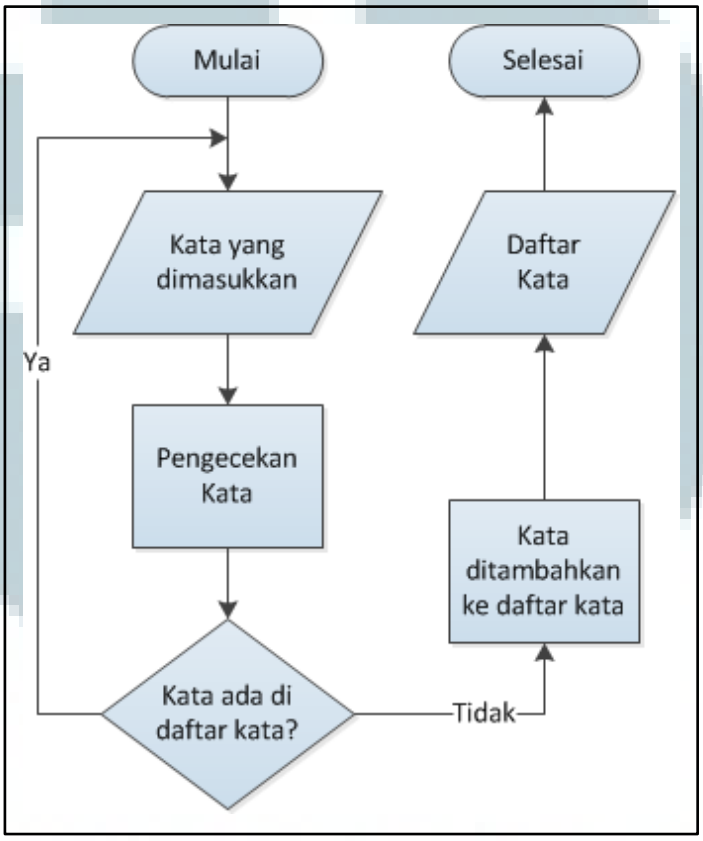


Gambar 3.12 *Flowchart* Perhitungan Nilai *Similarity*

Setelah pencocokan selesai dilakukan dan nilai ketemu sudah ditentukan, aplikasi akan menghitung nilai *similarity* atau persentase kemiripan dokumen menggunakan rumus *Dice's coefficient* yang sudah dijelaskan pada bab

sebelumnya dan menampilkannya pada aplikasi. Perhitungan nilai *similarity* dilakukan dengan nilai ketemu dibagi dengan jumlah dari *n-grams* dokumen asli dan *n-grams* dokumen uji dan dikali dengan seratus.

3.4 Fitur Lainnya



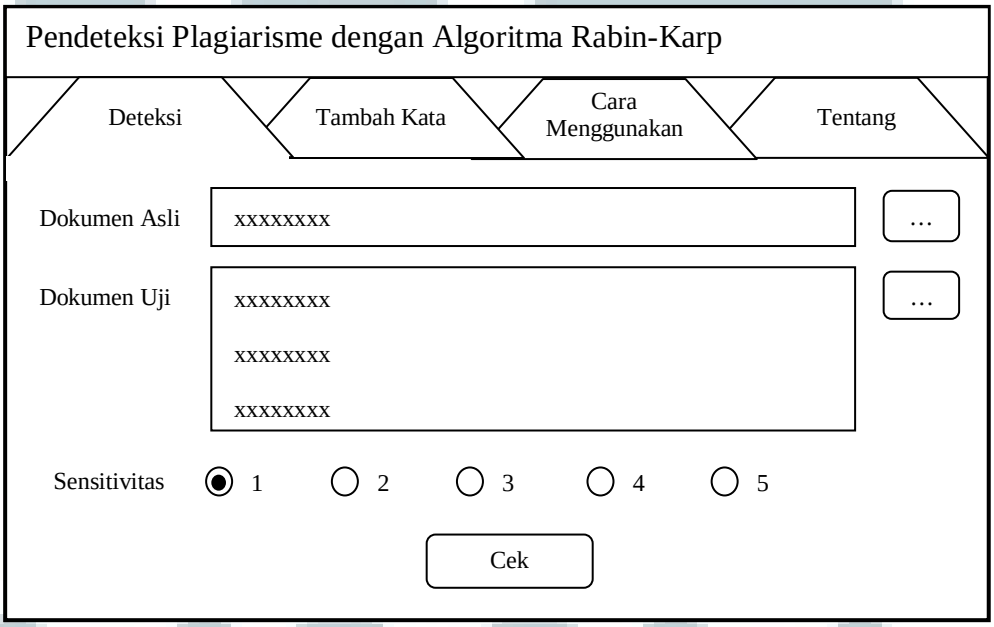
Gambar 3.13 *Flowchart* Modul Penambahan Kata

Selain modul untuk mendeteksi plagiarisme, aplikasi ini juga memiliki modul untuk menambahkan kata baru ke dalam kamus atau *stop word list.* Tujuannya adalah bisa meningkatkan keakuratan dalam tahap *filtering* dan *stemming.* Dalam modul ini, *user* dapat memasukkan kata baru dan memasukkannya ke dalam kamus atau *stop word list*. Jika kata yang ingin

dimasukkan sudah ada dalam daftar kata,kata yang baru tidak bisa ditambahkan. Jika tidak ada, kata baru bisa ditambahkan ke dalam daftar kata.

3.5 Desain Antarmuka Aplikasi

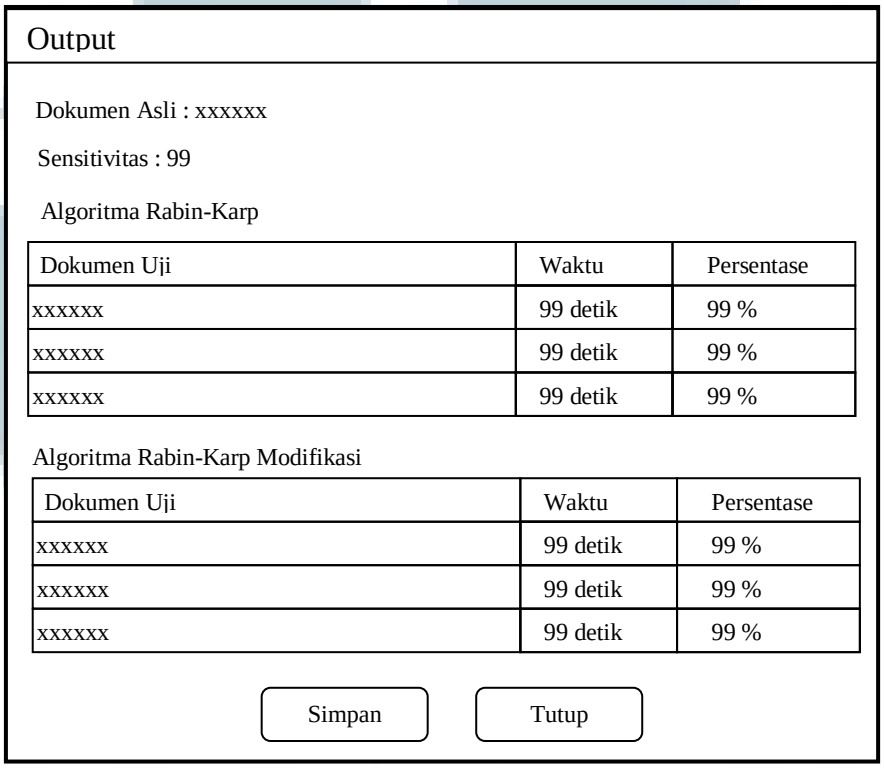
Desain antarmuka aplikasi merupakan rancangan tampilan dari aplikasi. Desain ini akan digunakan sebagai panduan dalam membangun antarmuka yang akan dilihat *user* untuk menggunakan aplikasi. Berikut adalah desain antarmuka dari aplikasi pendeteksi plagiarisme.



Gambar 3.14 Desain Antarmuka Aplikasi Pendeteksi Plagiarisme

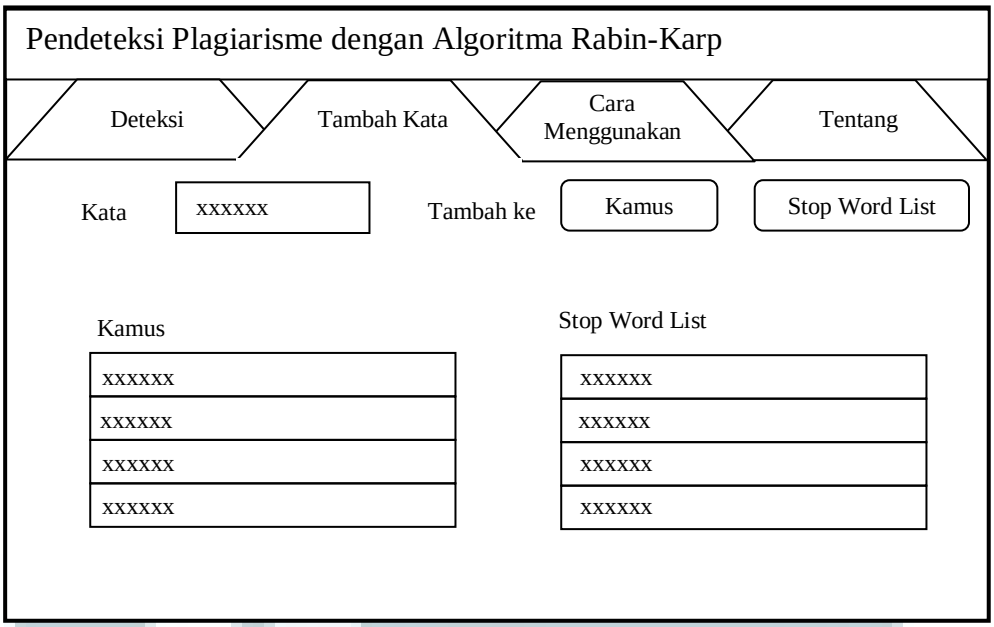
Gambar di atas merupakan tampilan yang akan dilihat *user* ketika aplikasi dibuka. Aplikasi ini memiliki empat buah *tab. Tab* pertama berisikan komponen-komponen yang diperlukan untuk mendeteksi plagiarisme pada dokumen teks. *Tab* ini memiliki tiga buah tombol. Tombol dengan tulisan "..." berfungsi untuk memilih dokumen asli dan dokumen uji yang ingin dibandingkan. Ketika ditekan, akan muncul sebuah *window* untuk memilih *file*. Tombol berikutnya adalah

tombol “Cek” yang berfungsi untuk memulai proses pendeteksian plagiarisme pada dokumen teks. Selain itu, terdapat lima buah *radio button*yang digunakan untuk menentukan nilai dari *n-grams.* Berdasarkan kegunaannya, *tab* ini akan dinamakan “Deteksi”. Gambar 3.14 menunjukkan tampilan dari *tab* “Deteksi”.



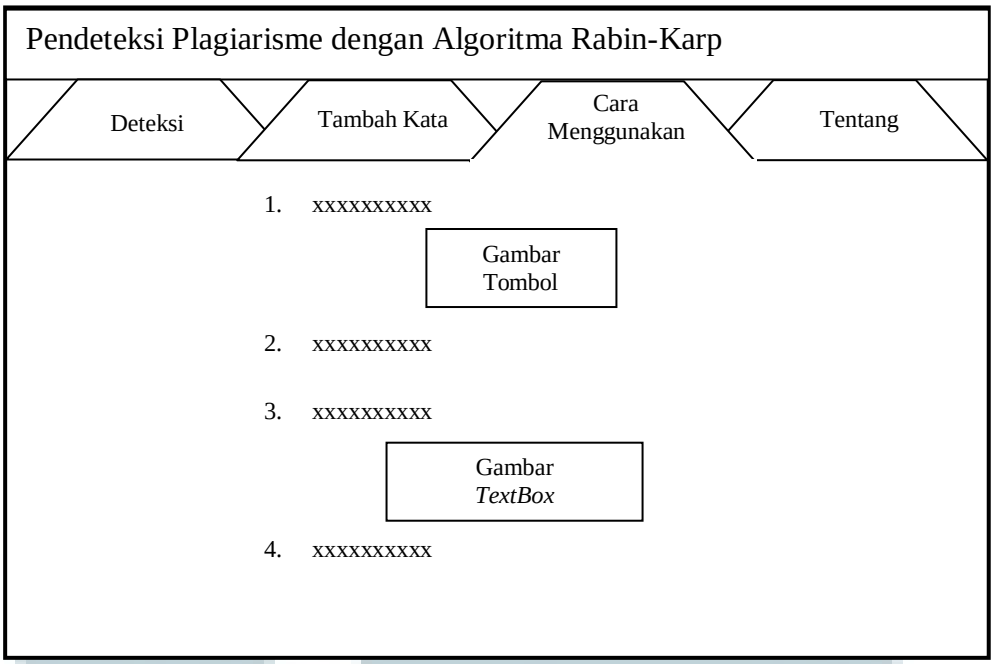
Gambar 3.15 Desain Antarmuka *Output Window*

Ketika tombol “Cek” ditekan, proses pencocokan akan dilakukan dan *output* akan ditampilkan dalam sebuah *window* baru. Dalam *window* ini, terdapat dua *list view* yang menampilkan seluruh perhitungan nilai kemiripan dan waktu proses algoritma Rabin-Karp dan yang dimodifikasi dari setiap dokumen uji. Di samping itu, terdapat dua buah tombol, yaitu tombol “Tutup” yang berfungsi untuk menutup *window* dan “Simpan” yang berfungsi untuk menyimpan *output* ke dalam *text file*.



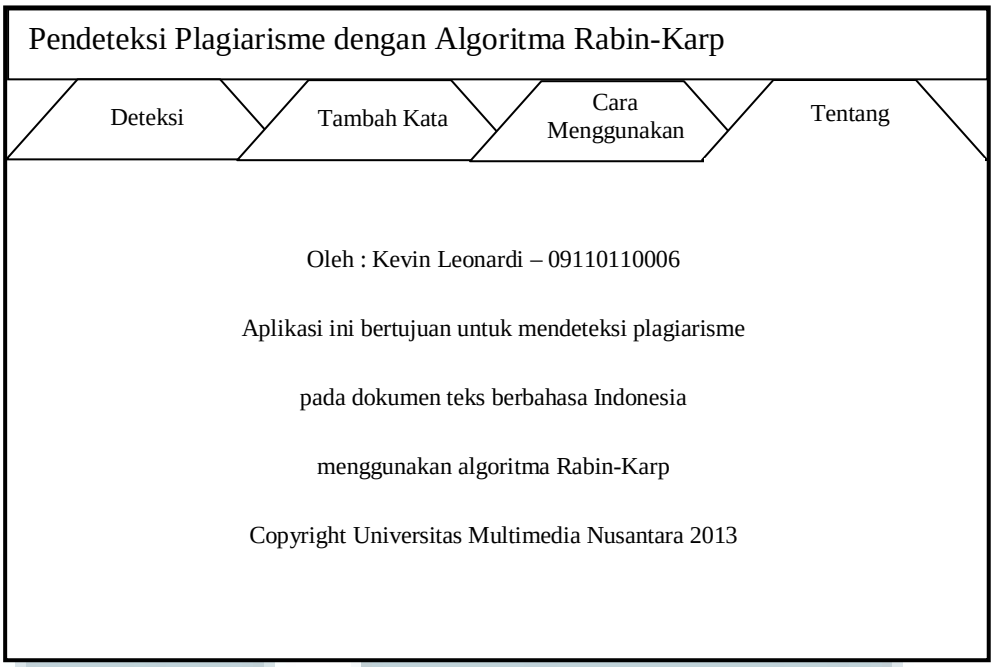
Gambar 3.16 Desain Antarmuka *Tab* "Tambah Kata"

Tab selanjutnya digunakan untuk menampung komponen-komponen yang dibutuhkan untuk melakukan modul penambahan kata. Dalam *tab* ini, terdapat dua buah *list view* yaitu *list view* kamus dan *list view stop word list* yang berguna untuk menampilkan seluruh daftar kata yang ada, sebuah *text box* untuk mengetikkan kata, dan dua buah tombol untuk menambahkan kata ke daftar kata yang sesuai dengan nama tombol. Kedua tombol ini bisa ditekan jika kata yang diketikkan tidak ditemukan dalam *list view.* Berdasarkan kegunaannya, *tab* ini akan dinamakan "Tambah Kata".



Gambar 3.17 Desain Antarmuka *Tab* "Cara Menggunakan"

Tab selanjutnya berfungsi untuk menunjukkan bagaimana cara menggunakan aplikasi dan dinamakan "Cara Menggunakan". Penjelasan meliputi bagaimana cara untuk melakukan pendeteksian plagiarisme dan menambahkan kata baru. Penjelasan dipaparkan dalam langkah-langkah dan ditampilkan dalam bentuk tulisan dan gambar. Desain antarmuka dari *tab* "Cara Menggunakan" digambarkan pada gambar 3.17.



Gambar 3.18 Desain Antarmuka *Tab* "Tentang"

Tab terakhir dalam aplikasi berisikan teks yang bertuliskan nama pembuat aplikasi, tujuan dari aplikasi, tahun pembuatan aplikasi, dan hak cipta aplikasi. Tujuannya adalah agar *user* mengetahui apa kegunaan dari aplikasi ini. *Tab* ini dinamakan "Tentang".