BAB III
METODE PENELITIAN DAN PERANCANGAN

Dalam bab ini dibahas perancangan sistem deteksi plagiarisme mengguna-kan algoritma Rabin-Karp dengan *synonym recognition*. berikut penjelasan langkah-langkahnya:

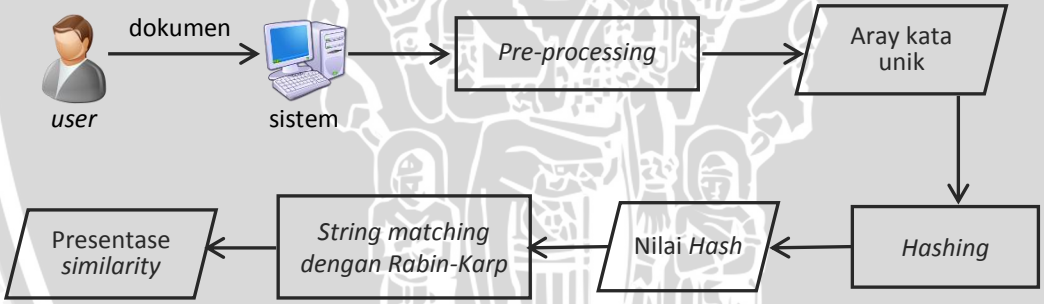
1. Mendalami konsep algoritma Rabin-Karp beserta metode-metode pendukungnya yang digunakan untuk mencari kecocokkan *string*.
2. Melakukan analisa dan perancangan sistem deteksi plagiarisme menggunakan algoritma Rabin-Karp dengan *synonym recognition*.
3. Mengimplementasikan sistem berdasarkan analisa dan perancangan sebelumnya.
4. Melakukan uji coba terhadap sistem yang telah dibuat dengan menganalisa output sistem yaitu waktu proses dan presentase kesamaan *string* antar dokumen, dimana output juga dipengaruhi penggunaan *synonym recognition.*
5. Mengevaluasi output (waktu proses dan presentase kesamaan) pencocokan *string* algoritma Rabin-Karp dengan dan tanpa *synonym recognition.*

3.1. Data Penelitian

Dalam penelitian ini sistem deteksi kesamaan dokumen menggunakan data jurnal berbahasa Indonesia yang diambil dari kumpulan jurnal JTIF PTIIK. Dari dokumen teks yang diperoleh dilakukan seleksi fitur dengan hanya mengambil bagian pendahuluan sebagai sebagai *sample* yang akan diproses. Dikarenakan tiap dokumen memiliki tema dan judul yang beragam, dari tiap judul dokumen akan dimanipulasi sedemikian rupa sehingga menghasilkan dokumen serupa dengan kesamaan sekitar 80%, 60%, 40% dan 20%. Selain mengganti kata dengan kata lain yang memiliki makna sama, manipulasi dokumen juga termasuk mengubah struktur kalimat (parafrase). Dari varian dokumen inilah yang akan dibandingkan dan diperiksa kesamaannya dengan dokumen yang asli.

3.2. Deskripsi Umum Sistem

Sistem deteksi plagiarisme ini secara umum bekerja dengan membanding-kan dokumen asli dengan dokumen yang ingin diuji dan mencari kecocokan antar *string*-nya. Dari presentase banyaknya kesamaan inilah dijadikan indikator untuk menentukan dokumen tersebut plagiat atau tidak. Diawali dengan *user* menginputkan dua dokumen teks yang akan dibandingkan. Sebelum memulai proses *user* diberi opsi apakah menggunakan *synonym recognition* atau tidak. *User* juga menentukan nilai parameter *k-gram* dan modulo. Proses *pre-processing* akan membersihkan data dari noise seperti tanda baca (karakter selain huruf dan angka), dari kata sambung, dan men-*set* semua *string* ke dalam bentuk *lower case*. Kemudian sistem akan mencari kecocokan antar dokumen teks dengan algoritma Rabin-Karp dan menghasilkan output presentase kesamaan *string* berikut waktu prosesnya. Berikut penggambaran skema aliran data dalam sistem deteksi plagiarisme:



Gambar 3.1. Skema aliran data.

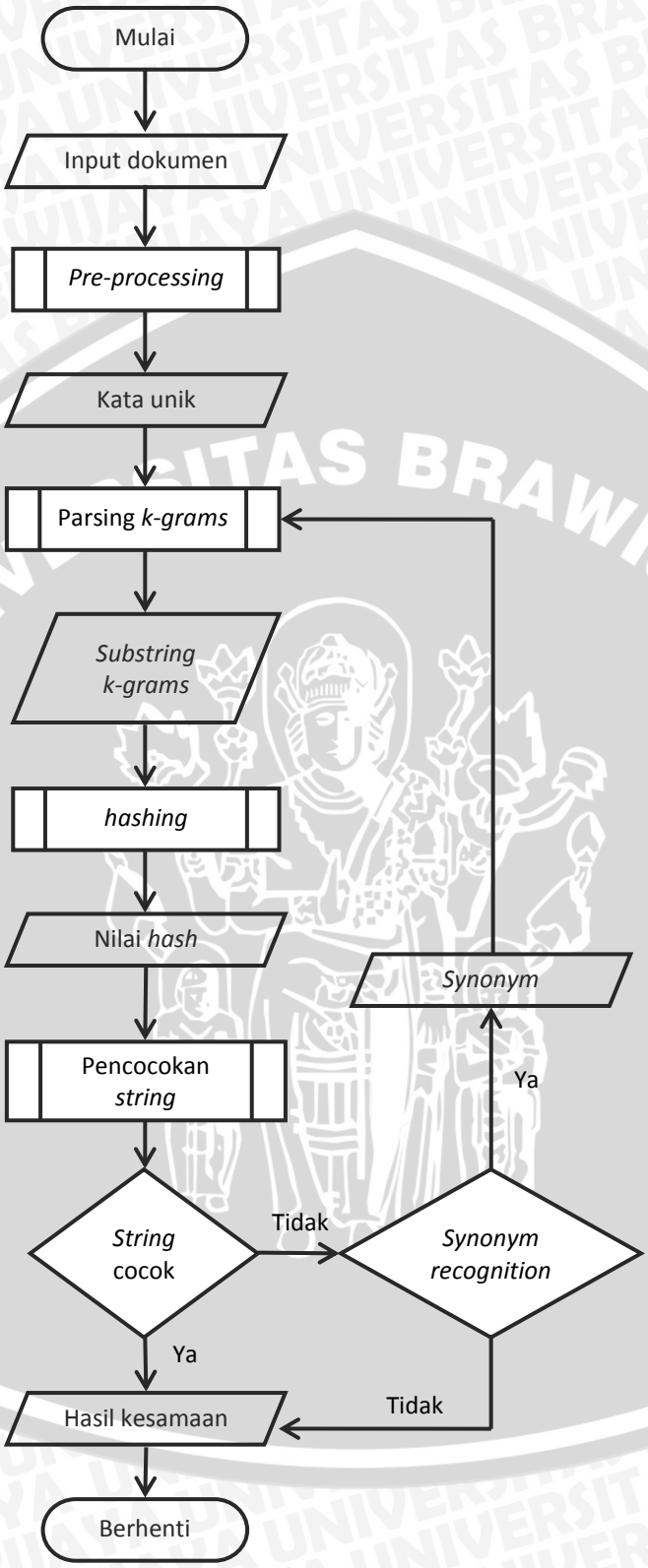
Pada Gambar 3.1 terlihat *user* memasukkan data berupa dokumen teks yang berupa *file notepad* (*.txt), dan *Microsoft word* (*.doc, dan *.docx). Di dalam sistem dilakukan *pre-processing* seperti *case folding*, *tokenizing*, dan *filtering*. Setelah *pre-processing*, data akan diproses menggunakan algoritma Rabin-Karp. Algoritma Rabin-Karp akan mencocokkan dokumen dengan mencari kesamaan *string*. Alih-alih mencocokkan *string* secara langsung algoritma Rabin-Karp mencocokkan nilai *hash*.

Nilai *hash* diperoleh dari *string* unik hasil *pre-processing* yang sudah diurai menjadi *substring k-gram* dan kemudian di-*hash* menggunakan fungsi *hash*. nilai *hash* inilah yang merepresentasikan *string* dan dicocokkan antar dokumen menggunakan algoritma Rabin-Karp. Jumlah *string* yang sama akan dihitung dengan koefisien kesamaan Dice (*Dice's similarity coefficient*) untuk menghasilkan presentase kesamaan.

3.3. Perancangan Proses

Proses dalam sistem deteksi plagiarisme dalam penelitian ini dapat digolongkan ke dalam dua proses inti yaitu *pre-processing* dan pencocokan *string*. Pada fase *pre-processing* sistem membaca dokumen inputan yaitu dokumen asli dan dokumen yang akan diuji. Dokumen teks ini kemudian diubah ke dalam bentuk *lower case*, dihilangkan karakter selain huruf dan angka dan ditampilkan dalam bentuk *token* per kata. Fase pencocokan *string* ini dimulai dengan mengurai kata unik menjadi *substring* sepanjang *k-gram* tergantung masukan *user*. Menggunakan fungsi *hash* yang sudah dijabarkan dalam Bab 2, dan dengan parameter modulo yang juga diinputkan *user*, *substring k-gram* dicari nilai *hash*-nya. Sehingga dihasilkan keluaran *remain* (sisa hasil bagi modulo) dan *quotient* (hasil bagi modulo) dari tiap sub*string* dalam dokumen teks. Kedua poin inilah yang akan dibandingkan sistem untuk menemukan kesamaan *string* antar dokumen.

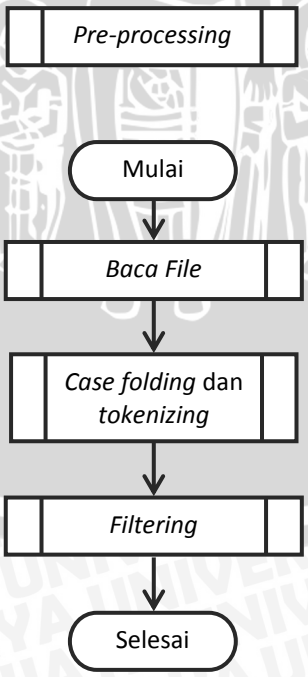
Bila *user* memilih opsi fitur *synonym recognition*, *string* hasil pencocokan yang tidak sama akan dicari ke dalam *database* untuk menggantinya dengan *synonym*. *Synonym* (kata lain) hasil pergantian ini akan dilakukan *pre-processing* dan diproses kembali melalui fase pencocokan *string*, sampai tidak lagi ditemukan *string* yang sama. Hasil akhir *string* yang sama akan dikonversi ke dalam presentase kesamaan. Perancangan proses secara global ditunjukkan dalam bentuk *flowchart* seperti pada Gambar 3.2.



Gambar 3.2. *Flowchart* sistem secara global.

3.3.1. Perancangan *Pre-processing*

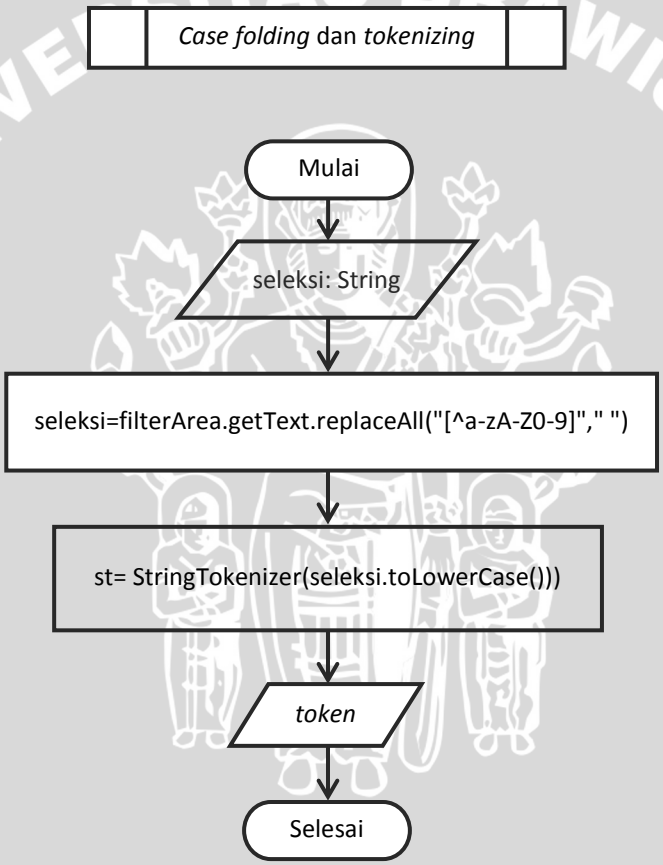
Pre-processing menangani input dokumen teks yang diinputkan oleh *user* untuk kemudian diolah pada tahap pencocokan *string*. Sistem membaca *file* masukkan berupa *file* .txt, doc, atau docx yang kemudian akan menampilkan teks dalam dalam pada teks area. Diasumsikan dokumen teks adalah data teks yang mengandung banyak *noise* berupa huruf kapital dan tanda baca. Karena itu dilakukan *case folding* untuk merubah semua kata menjadi rata bawah (*lower case*).Karena yang dicocokkan adalah *string* dilakukan proses *tokenizing* untuk memisahkan setiap kata dalam dokumen teks menjadi *token string* per kata. Di sini karakter selain huruf, angka dan spasi tidak dipakai sehingga dihilangkan. Demi mendapatkan dokumen teks yang lebih ramping dilakukan proses *filtering* dengan menghapus kata yang tidak memiliki makna maupun kata sambung. Dokumen teks menghapus kata yang sesuai dengan daftar kata tidak bermakna (*stopword*) yang ditampung dalam *file*stopword Indonesia.txt. Untuk lebih lengkapnya isi stopword Indonesia.txt ditampilkan di Lampiran I. *Pre-processing* digambarkan melalui bagan alur seperti ditunjukkan pada Gambar 3.3.



Gambar 3.3. *Flowchart* fase *pre-processing*.

3.3.1.1.Perancangan *Case Folding* dan *Tokenizing*

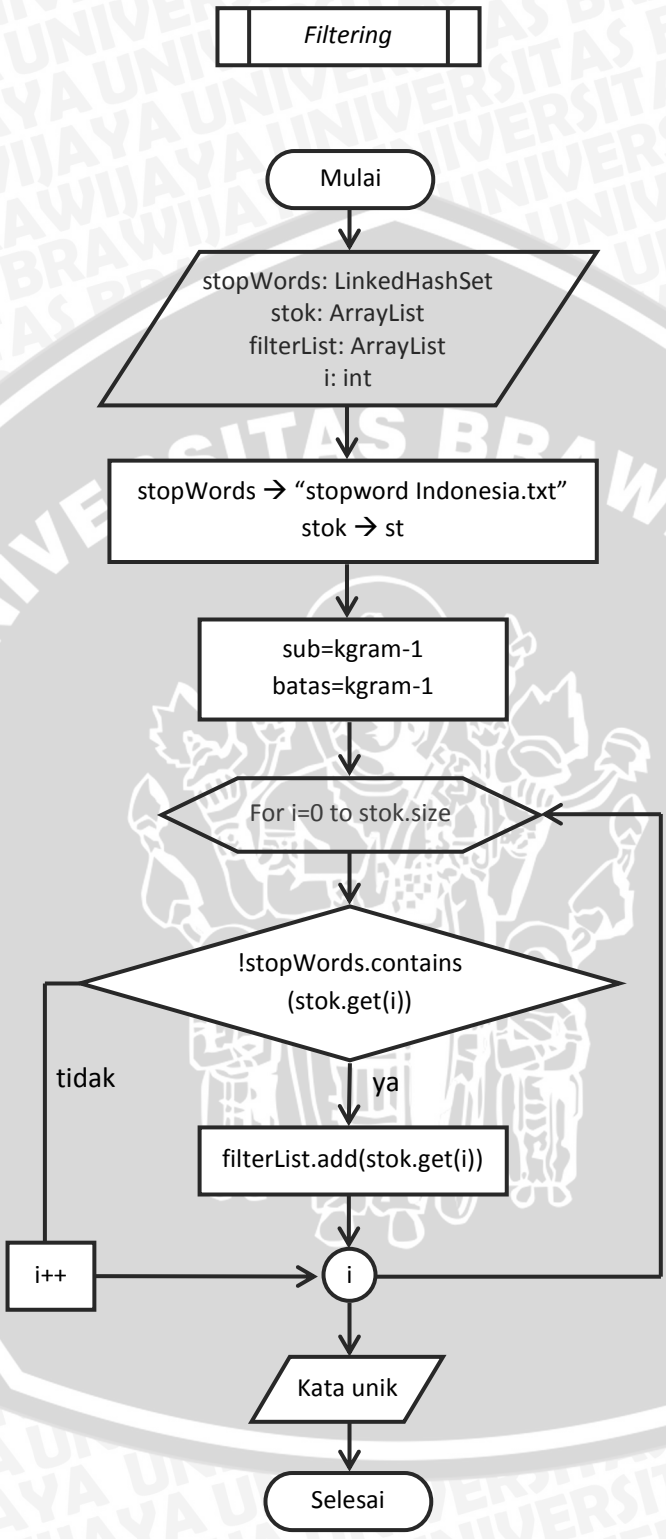
Data teks yang telah diekstrak dari *file* kemudian diproses di dalam *case folding* dan *tokenizing*. Pertama hasil ekstraksi dokumen teks ditampung ke dalam *string* seleksi. Teks kemudian diubah ke dalam huruf kecil dan dihilangkan karakter selain huruf dan angka. Karena yang akan ditangani adalah *string* (kata) maka dokumen dipecah menjadi *token-token*. Kata hasil pembersihan diubah ke dalam *token string* st. Gambaran mengenai proses *case folding* dan *tokenizing* dijelaskan melalui *flowchart* seperti pada Gambar 3.4.



Gambar 3.4. *Flowchart case folding* dan *tokenizing*.

3.3.1.2. Perancangan *Filtering*

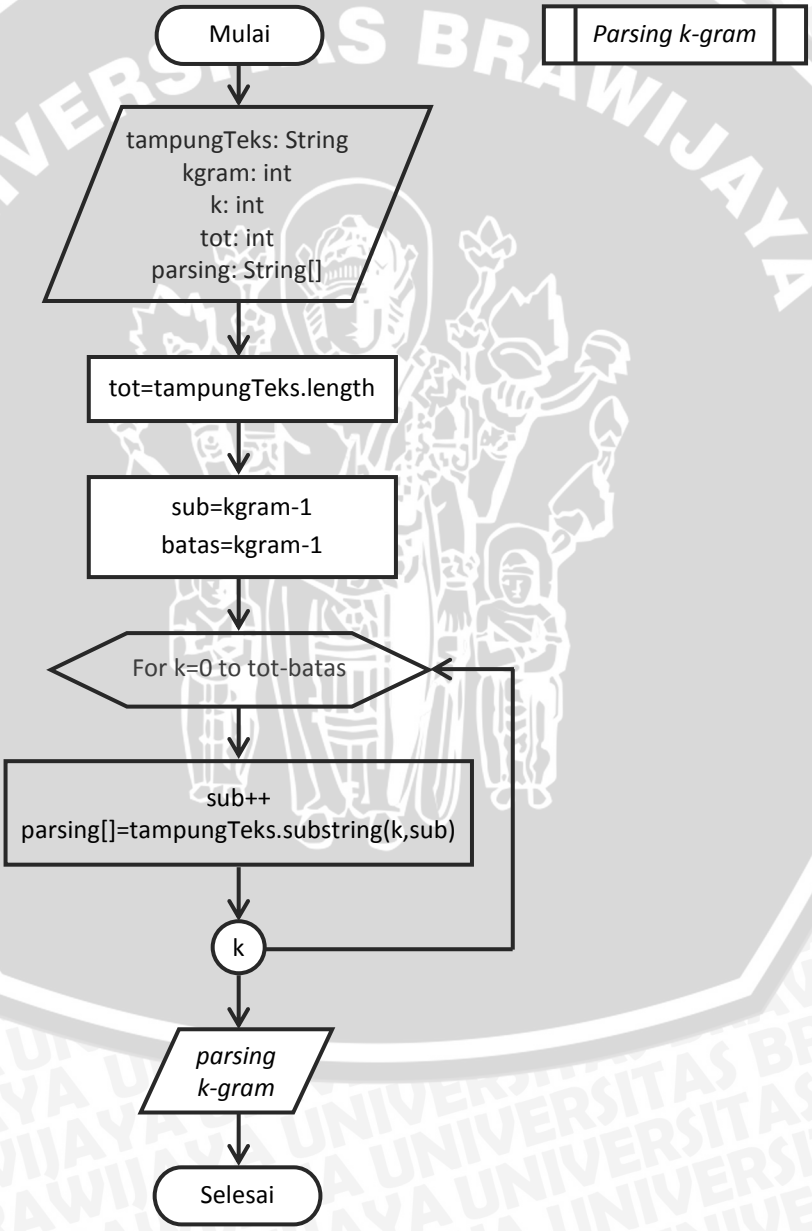
Selanjutnya sistem menghilangkan kata yang tidak memiliki makna dengan mencocokkan pada *file* stopword Indonesia.txt. Jika kata tidak terdapat pada daftar kata stopword Indonesia.txt kata akan dimasukkan dalam filterList. Prosesnya ditunjukkan pada Gambar 3.5.



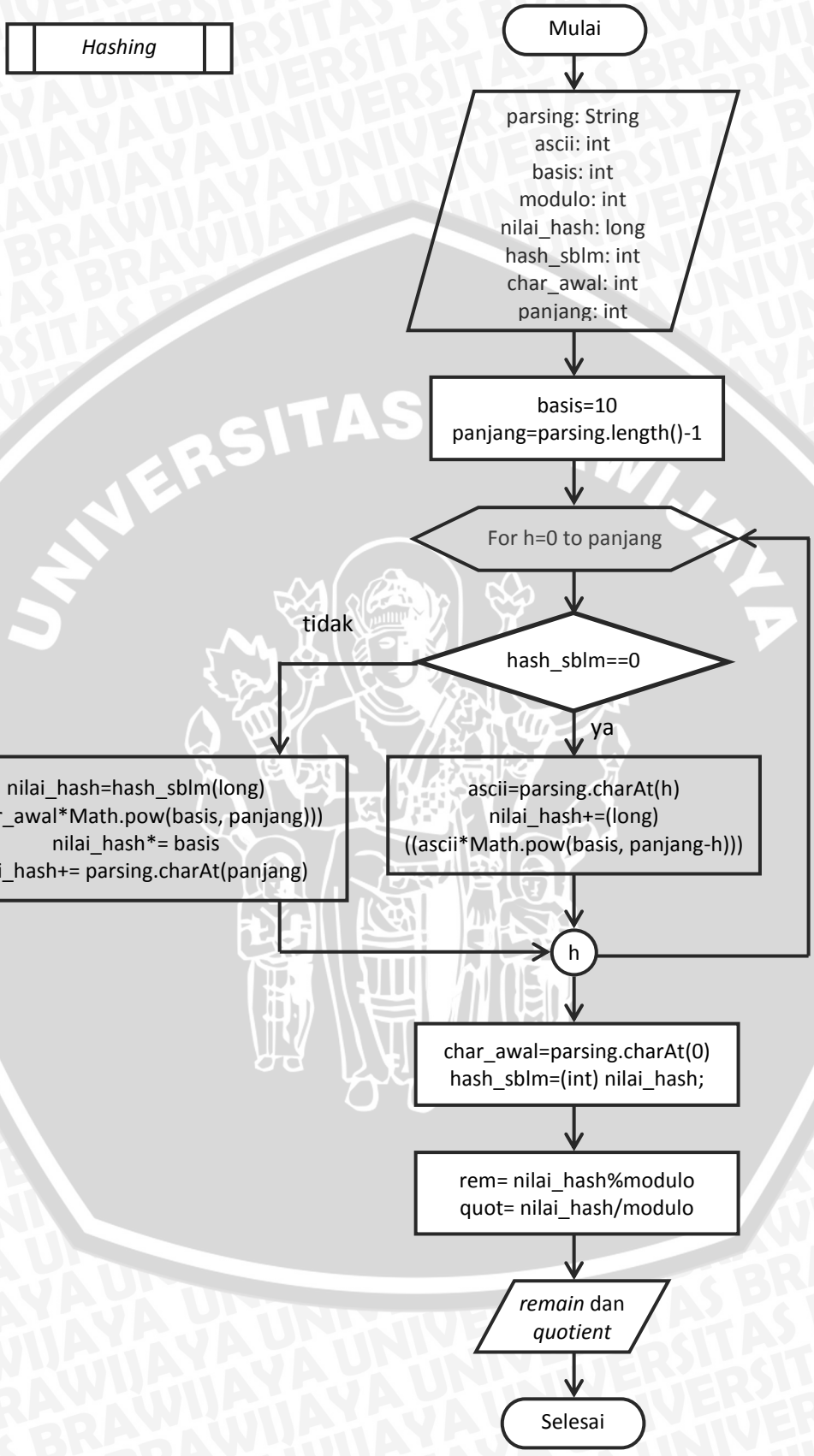
Gambar 3.5. *Flowchart filtering*

3.3.2. Perancangan Proses *Parsing K-gram*

Selepas melewati *pre-processing* data teks siap untuk diproses ke dalam pencocokan *string*. Tapi sebelumnya data teks harus diurai terlebih dulu menjadi *substring k-gram*. Panjangnya sub*string k-gram* tergantung inputan *k-gram* oleh *user* dengan cakupan 1 sampai 5. Hasilnya adalah rangkaian sub*string k-gram* yang berurutan. Proses lebih jelasnya digambarkan dalam *flowchart* dalam Gambar 3.6.



Gambar 3.6. *Flowchart* proses *parsing k-gram*



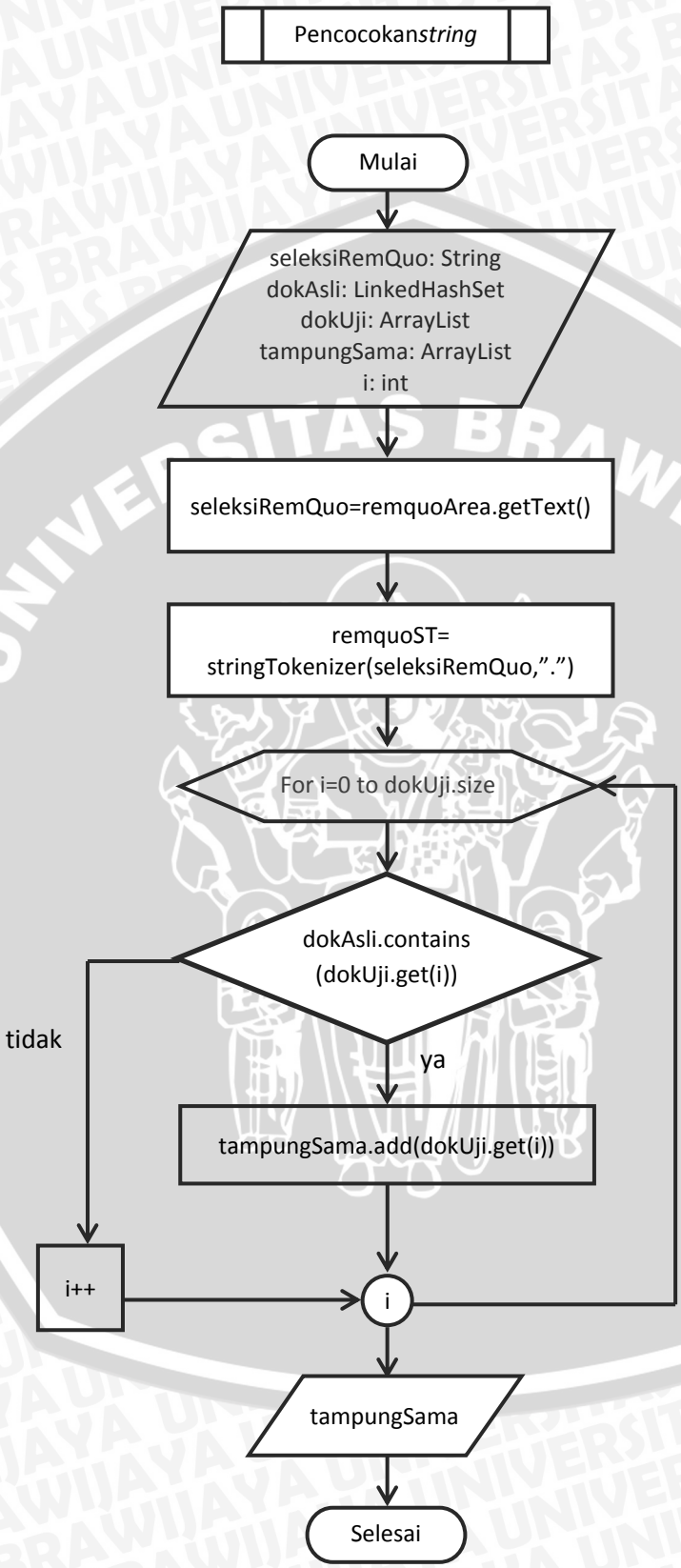
Gambar 3.7. *Flowchart hashing.*

3.3.3. Perancangan Proses *Hashing*

Pada Gambar 3.7 tampak proses hashing pada sistem. Proses *hashing* dimulai dengan menginisiasi nilai untuk basis yaitu 10. Nilai 10 dipilih karena merupakan banyaknya varian untuk angka (‘0’-‘9’). Dokumen teks yang telah melalui *pre-processing* dan sudah dilakukan *parsing k-gram* dimasukkan ke dalam *string parsing.* Untuk tiap *string* kata dalam kedua dokumen (dokumen asli dan dokumen uji) akan diubah ke dalam nilai *hash* melalui fungsi *hashing*. Eksekusi fungsi *hash* melihat dari nilai *hash* awal (variabel hash_sblm). Jika hash_sblm = 0 maka itu adalah *string* pertama. Jika didapat *string* pertama dilakukan perhitungan nilai *hashing string* pertama menggunakan persamaan fungsi *hash* seperti pada persamaan (2-1) di Bab 2. Sistem menghitung nilai hash sebuah *string* dengan mengalikan nilai ASCII tiap karakter dengan basis berpangkat panjang *string* dikurangi 1. Nilai hash yang dihasilkan kembali diinisiasi ke dalam hash_sblm. Sedangkan untuk *string-string* berikutnya dilakukan perhitungan dengan *rolling hash. Rolling hash* mendapatkan nilai *hash* dengan menghitung dari nilai *hash* yang sebelumnya. Untuk mendapatkan nilai *hash* diambil dua komponen yaitu *rem* (*remain*) dan *quot* (quotient) yang merupakan hasil variabel nilai_hash di-mod oleh modulo dan nilai_hash dibagi dengan modulo.

3.3.4. Perancangan Proses Pencocokan *String*

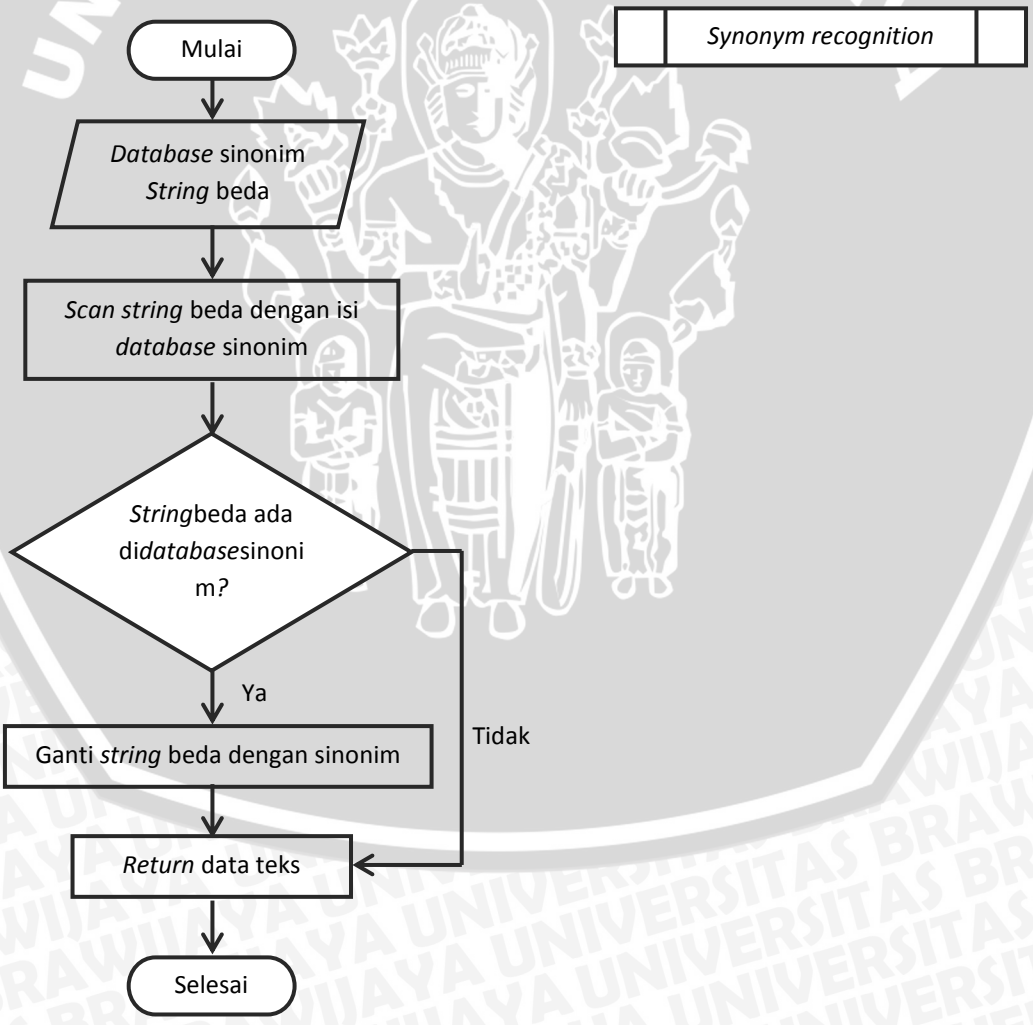
Pada proses pencocokan *string*, sistem membaca hasil *rem* dan *quo* yang telah ditampung di variabel *string* seleksiRemQuo. Sebanyak *substring* dokumen uji dilakukan pembandingan nilai *hash*. Jika *rem* dan *quo* dokumen asli sama dengan yang ada di dalam dokumen uji, string yang sama dari dokumen uji dimasukkan ke dalam *ArrayList* tampungSama. Jika tidak akan berlanjut ke *string* berikutnya. Karena sistem langsung membandingkan *remain* dan *quotient* secara bersamaan maka diasumsikan tidak ada *spurious hit*. Karena itu tidak perlu dilakukan pengecekan terhadap setiap karakter dalam *substring* untuk memastikan *string* cocok atau tidak. Proses kerja pencocokan *string* ditunjukkan dalam Gambar 3.8.



Gambar 3.8. *Flowchart* pencocokan *string*.

3.3.5. Perancangan Proses *Synonym Recognition*

Setelah proses pencocokan *string* ada kemungkinan ditemui beberapa *string* yang tidak sama. Melalui *synonym recognition* sistem mencari kata ganti dari *string* yang berbeda dalam *database* sinonim. *Database* sinonim ber-*platform* mySQL dan memiliki maksimal tiga kata sinonim. Jika *string* (kata) tersebut terdapat di *database* sinonim maka akan diganti dengan kata sinonimnya. Jika tidak ada, *string* tetap dan tidak diganti. Data teks setelah pergantian ini kemudian diproses kembali dalam *pre-processing* dan pencocokan *string*. Jika kata yang sudah diganti tetap tidak cocok dengan yang ada di dokumen asli, *synonym recognition* terus dilakukan sampai tidak ada lagi kesamaan atau tidak ada *string* beda. Prosesnya bisa terlihat dalam Gambar 3.9.



Gambar 3.9. *Flowchart synonym recognition*

3.4.Perhitungan Manual

Perhitungan manual mengambil dokumen xDeny-ori.docx (Dokumen1) dan xDeny-x1.docx (Dokumen2) untuk dibandingkan. Dokumen xDeny-ori.docx merupakan penggalan dari *file* sample dari dokumen uji Deny-ori.docx yang dipakai dengan hanya mengambil kalimat pertamanya saja. Dan dokumen xDeny-x1.docx merupakan hasil modifikasi.

Dokumen1: "Perkembangan teknologi informasi yang pesat dan semakin besar pemanfaatan teknologi komputer di berbagai bidang membuat kebutuhan akan pengelolaan data, seperti data teks, citra, audio dan video semakin meningkat pula."
Dokumen2: "Di tengah kemajuan teknologi informasi yang tumbuh pesat dan semakin besarnya pendayagunaan teknologi komputer di semua bidang membuat kebutuhan pengelolaan data, seperti data teks, citra, audio dan video semakin meningkat juga."

Melalui tahap *pre-processing* (*case folding*, *tokenizing*, dan *filtering*) dihasilkan *substring* kata dasar unik:
Dokumen1: "kembang teknologi informasi pesat manfaat teknologi komputer bidang butuh kelola data data teks citra audio video tingkat".
Dokumen2: "kembang maju teknologi informasi tumbuh pesat dayaguna teknologi komputer bidang butuh kelola data data teks citra audio video tingkat".

Setelah dilakukan *parsing k-gram* dengannilai *k-gram* = 4 dihasilkan sub*string* sebagai berikut:
Dokumen1: "kemb emba mban bang angt ngte gtek tekn ekno knol nolo olog logi ... audi uido idov dovi ovid vide ideo deot eoti otin ting ingk ngka gkat".
Jumlah *substring k-gram* = total karakter tanpa spasi - (k-1) (3-1)
Total *substring* 4-gram: 106.

Dokumen2: "kemb emba mban bang angm ngma gmaj maju ajut jutt utte ttek tekn ekno knol nolo olog logi ... vide ideo deot eoti otin ting ingk ngka gkat".
Total sub*string* 4-gram: 113.

Dengan basis = 10 dan modulo = 101, dilakukan perhitungan nilai *hash* sesuai dengan persamaan (3-2).

$Hash(w[0...m\text{-}1]) = (w[0]*b^{m-1} + w[1]*b^{m-2} + ... + w[m\text{-}1]*b^{0}) \bmod q$ (3-2)

Remain atau sisa ($h1/q$) = sisa ($h2/q$) (3-3)

Quotient atau hasil bagi ($h1/q$) = hasil bagi ($h2/q$) (3-4)

"kemb" $= (107*10^3) + (101*10^2) + (109*10^1) + (98*10^0)$

$= 107000 + 10100 + 1090 + 98$

$= 118288$

Remain (sisa modulus) = 118288 mod 101 = 17

Quotient (hasil bagi) = 118288/101 = 1171,1683

Untuk perhitungan *hash* berikutnya dilakukan *rehash* dengan metode *rolling hash menurut persamaan (3-5)*:

$Rehash(x,y,hash) = ((hash - x*b^{m-1})*b + y) \bmod q$ (3-5)

"emba" $= (118288-(107*10^3))*10+97$

$= (118288-107000)*10+97$

$=112880 + 97 = 112977$

Rem = 112977 mod 101 = 59

Quo = 112977/101 = 1118,5841

"mban" $= (112977-(101*10^3))*10+110$

$= (112977-101000)*10+110$

$= 119770 + 110 = 119880$

Rem = 119880 mod 101 = 94

Quo = 119880/101 = 1186,9307

....

Setiap nilai *hash substring* (*remain* dan *quotient*) dicocokkan dengan *set* (himpunan) nilai *hash* pola, hasilnya seperti ditunjukkan dalam tabel 3.1.

Tabel 3.1 Pencocokan *string*

No	Dokumen 1 (teks)			No	Dokumen 2 (pola)			*String* Cocok
	Substring	*Rem*	*Quo*		*Substring*	*Rem*	*Quo*	
1	kemb	17	1171,1683	1	kemb	17	1171,1683	Ya
...	...	...	...	...	...	...	...	...
5	angt	66	1080,6535	5	angm	59	1080,5844	Tidak
6	ngte	58	1203,5743	6	ngma	54	1203,5347	Tidak
...	...	..	...	...	...	...	...	...
105	ngka	65	1202,6436	112	ngka	65	1202,6436	Ya
106	gkat	50	1136,4950	113	gkat	50	1136,4950	Ya

Dari hasil pencocokan *string*, dimana sistem tidak menggunakan fitur *synonym recognition* dihasilkan presentase kesamaan sementara:

Substring k-gram unik dokumen asli (A): 102

Substring k-gram unik dokumen uji (B): 113

Jumlah *substring* sama (C): 87

Presentase kesamaan = 2C/(A+B)

= ((2*87)/(102+113))*100%

= (174/215)*100%

= 80,93%

Selain itu didapati bahwa *string* yang tidak sama adalah: "maju, tumbuh ,dan daya guna". Jika sistem memakai *syonym recognition* kata tersebut digantikan dengan "berkembang, maju, manfaat". Sehingga dokumen uji menjadi:

Dokumen2: "kembang berkembang teknologi informasi maju pesat manfaat teknologi komputer bidang butuh kelola data data teks citra audio video tingkat".

Setelah kembali melalui *pre-processing* dan diubah menjadi *substring*:

Dokumen2: "kemb emba mban bang angk ngke gkem kemb emba mban bang angm ngma gmaj maju ... citr itra traa raau aaud audi udio diov iovi ovid vide ideo deot eoti otin ting ingk ngka gkat".

Total *substring* 4-gram: 113.

Data teks dokumen2 setelah dijadikan *parsing k-gram* diubah ke dalam nilai *hash* melalui fungsi *hash* yang sama. Hasil *remain* dan *quotient* dari dokumen2 kembali dilakukan pencocokan *string* dengan dokumen1. Proses pencocokan secara manual dapat dilihat pada tabel 3.2.

Tabel 3.2 Pencocokan *string*setelah *synonym recognition.*

No	Dokumen 1 (teks)			No	Dokumen 2 (pola)			*String*
	Substring	*Rem*	*Quo*		*Substring*	*Rem*	*Quo*	Cocok
1	kemb	17	1171,1683	1	kemb	17	1171,1683	Ya
...	...	...	...	...	...	...	...	...
5	angt	66	1080,6535	5	angk	59	1080,5844	Tidak
6	ngte	58	1203,5743	6	ngke	54	1203,5347	Tidak
...	...	..	...	...	...	...	...	...
28	satm	22	1247,2178	39	satm	22	1247,2178	Ya
29	atma	0	1087	40	atma	0	1087	Ya
30	tman	13	1267,1287	41	tman	13	1267,1287	Ya
...	...	..	...	...	...	...	...	...
105	ngka	65	1202,6436	112	ngka	65	1202,6436	Ya
106	gkat	50	1136,4950	113	gkat	50	1136,4950	Ya

Setelah dilakukan pencocokan *string* ulang dengan *k-gram* dan modulo yang sama diperoleh hasil:

Sub*string k-gram* unik dokumen asli (A): 102

Sub*string k-gram* unik dokumen uji (B): 113

Jumlah *substring* yang cocok (C): 99

Presentase kesamaan = 2C/(A+B)

= ((2*99)/(102+113))*100%

= (198/215)*100%

= 92,09%

3.5. Rancangan Uji Coba

3.5.1. Bahan Pengujian

Dari dokumen teks yang telah dikumpulkan, akan dipilih empat judul jurnal untuk mewakili data yang akan diuji. Tiap satu dokumen akan dibuat *copy* yang berupa dokumen yang diubah sedemikian rupa untuk menguji tingkat kesamaan. Bahan juga digunakan untuk menguji bagaimana perbedaan kinerja algoritma Rabin-Karp menggunakan *synonym recognition* dengan yang tidak.

3.5.2. Tujuan Pengujian

Pengujian sistem deteksi plagiarisme menggunakan algoritma Rabin-Karp bertujuan untuk:

1. Menganalisa peran parameter *k-gram* dalam proses *hashing* dilihat dari nilai kesamaan dan waktu proses.
2. Menganalisa presentase kesalahan (*error*) keluaran sistem yang menggunakan *synonym recognition* dan yang tidak menggunakan *synonym recognition* dengan parameter *k-gram* untuk validasi sistem.
3. Membandingkan hasil deteksi plagiarisme dokumen menggunakan algoritma Rabin-Karp dengan *synonym recognition* dan tanpa *synonym recognition* ditinjau dari presentase kesamaan kata dan waktu proses.

3.5.3. Skenario Pengujian

Pengujian yang akan dilakukan dalam sistem adalah dengan membandingkan dokumen asli dengan dokumen uji untuk menguji parameter *k-gram*. Hasil analisa akan menghasilkan *k-gram* ideal yang kemudian akan digunakan untuk menguji tingkat kesamaan dan waktu komputasi sistem baik yang menggunakan *synonym recognition* maupun yang tidak. Untuk menguji *k-gram* sistem membandingkan dokumen asli dengan satu dokumen yang sudah dimanipulasi 80% kesamaan kata. Untuk menguji pengaruh *synonym recognition* sistem membandingkan dokumen asli dengan beberapa dokumen yang sudah diubah ke dalam beberapa varian dengan tingkat kesamaan beragam.

3.5.4. Pengukuran Nilai Kesamaan

Karena sistem deteksi plagiarisme ini berpatok pada tingkat kesamaan kata maka nilai kesamaan dan presentase kesamaan kata menjadi indikator utama sebuah dokumen dikatakan plagiarisme atau tidak. Nilai dan presentase kesamaan didapatkan dari proses pembandingan nilai *hash* (*remain* dan *quotient*) antar dokumen. Dan sistem akan membandingkan nilai kesamaan jika memakai *synonym recognition* dengan yang tidak.

3.5.5. Pengukuran Waktu Proses

Sistem juga akan menghitung waktu proses yang dibutuhkan untuk menjalankan algoritma Rabin-Karp dalam mencocokkan *string*. Selanjutnya akan dianalisa perbandingan waktu proses algoritma Rabin-Karp jika menggunakan *synonym recognition* dengan tanpa *synonym recognition*.

3.5.6. Pengukuran Presentase Kesalahan (*Error*)

Presentase kesalahan (*error*) diperoleh dari hasil perbandingan nilai kesamaan yang dihasilkan oleh sistem diuji dengan nilai kesamaan yang diharapkan. *Substring* yang sama dari kedua dokumen akan dicocokkan kesamaan string yang sebenarnya apakah benar-benar sama. Jadi bisa dilihat apakah hasil keluaran sistem *valid* dan berjalan sesuai dengan yang diinginkan.

3.5.7. Perancangan Dokumen Asli dan Dokumen Uji

Dari empat judul jurnal yang telah dipilih dilakukan manipulasi teks secara manual untuk mendapatkan varian hasil dokumen uji seperti berikut:

a. 80% kesamaan. Sekitar sampai 20% kata akan diubah atau dihilangkan untuk mencapai 80% kesamaan.
b. 60% kesamaan. Perubahan dan pemotongan kata mencapai hingga 40% disertai dengan perubahan struktur kalimat.
c. 40%kesamaan. Lebih dari separuh kata memiliki makna lain dan beberapa dipotong untuk mendapatkan hingga 40% kesamaan.
d. 20% kesamaan. Dokumen diubah menjadi hampir bisa dikatakan berbeda dengan hanya sekitar 20% kesamaan kata.

Dokumen uji adalah hasil modifikasi dari dokumen asli jurnal sumber yang di-*edit* secara manual dengan memotong kata, memindah kata dan merubah struktur kalimat. Untuk memastikan apakah dokumen uji sesuai presentase kesamaan dilakukan pengecekan menggunakan tools pembandingan teks *online*. Bentuk dan rupa dokumen asli dan dokumen uji selengkapnya dapat dilihat di lampiran.

3.6. Perancangan Antarmuka

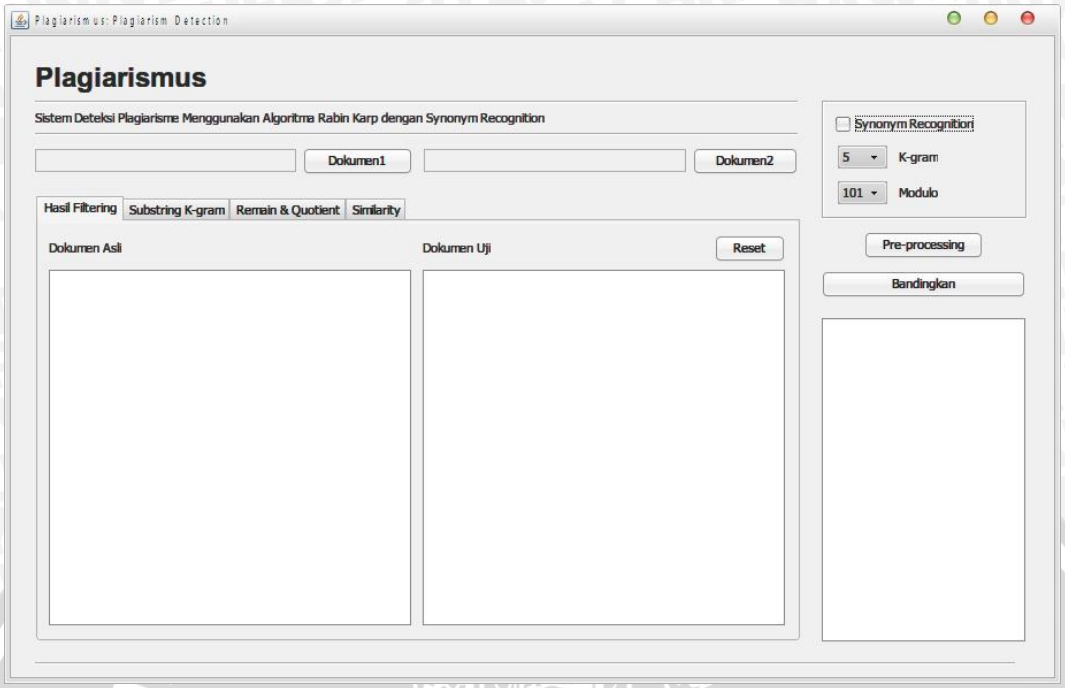
3.6.1. Perancangan Input Sistem

Sistem deteksi plagiarisme menggunakan algoritma Rabin-Karp dan *synonym recognition* memakai antarmuka GUI untuk memudahkan *user*. *User* bisa menginputkan *file* memilih opsi dan men-*set* parameter dalam interaksinya kepada antarmuka. Opsi dan parameter yang terdapat dalam sistem deteksi plagiarisme menggunakan algoritma Rabin-Karp dengan *synonym recognition* antara lain:

a. *File* sumber dokumen asli.
b. *File* sumber dokumen uji.
c. Opsi memakai *synonym recognition*.
d. Nilai parameter *k-gram*.
e. Nilai parameter modulo.

3.6.2. Perancangan *Protoype* Sistem

Antarmuka sistem dikembangkan melalui *platform* Java untuk penanganan program aplikasinya. Sistem juga menggunakan perangkat MySQL untuk menampung *database* kamus sinonim. Perangkat basis data MysSQL ini nantinya akan disambungkan ke dalam antarmuka untuk mengakomodasi beberapa proses. Berikut adalah tampilan desain antarmuka sistem deteksi plagiarisme menggunakan algoritma Rabin-Karp dan *synonym recognition* pada Gambar 3.10.



Gambar 3.10. Desain antarmuka sistem.