PERANCANGAN SISTEM DETEKSI PLAGIARISME DOKUMEN TEKS DENGAN MENGGUNAKAN ALGORITMA RABIN-KARP

SKRIPSI

Oleh:
Eko Nugroho
0510960022-96

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

PROGRAM STUDI ILMU KOMPUTER
JURUSAN MATEMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS BRAWIJAYA
MALANG
2011

LEMBAR PENGESAHAN SKRIPSI

PERANCANGAN SISTEM DETEKSI PLAGIARISME DOKUMEN TEKS DENGAN MENGGUNAKAN ALGORITMA RABIN-KARP

Oleh:
Eko Nugroho
0510960022-96

Setelah dipertahankan di depan Majelis Penguji
pada tanggal 11 Januari 2011
dan dinyatakan memenuhi syarat untuk memperoleh gelar Sarjana
dalam bidang Ilmu Komputer

Pembimbing I	Pembimbing II
Drs. Achmad Ridok, M.Kom.	Bayu Rahayudi, ST., MT
NIP. 19680807 199412 1 001	NIP. 19740712 200604 1 001

Mengetahui,
Ketua Jurusan Matematika
Fakultas MIPA Universitas Brawijaya
Ketua,

Dr. Agus Suryanto, MSc
NIP. 19690807 199412 1 001

LEMBAR PERNYATAAN

Saya yang bertanda tangan di bawah ini :

Nama : Eko Nugroho
NIM : 0510960022
Jurusan : Matematika
Penulis skripsi berjudul : Perancangan Sistem Deteksi Plagiarisme Dokumen Teks dengan Menggunakan Algoritma Rabin-Karp.

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, 11 Januari 2011
Yang menyatakan,

(Eko Nugroho)
NIM. 0510960022

PERANCANGAN SISTEM DETEKSI PLAGIARISME DOKUMEN TEKS DENGAN MENGGUNAKAN ALGORITMA RABIN-KARP

ABSTRAK

Plagiarisme merupakan tindakan menjiplak karya seseorang dan mengakuinya sebagai karya sendiri. Plagiarisme terhadap dokumen teks susah untuk dihindari. Oleh karena itu, sudah banyak diciptakan suatu sistem yang dapat membantu dalam melakukan deteksi palgiarisme dokumen teks seperti MOSS, TESSY, JPlag, CopyCatch dsb.

Untuk melakukan deteksi plagiarisme dokumen teks pada intinya adalah dengan melakukan pencocokan *string/terms*. Algoritma yang digunakan dalam skripsi ini adalah Rabin-Karp. Algortima ini digunakan karena cocok untuk pola pencarian jamak (*multiple pattern search*). Pada skripsi ini akan dilakukan sedikit modifikasi untuk meningkatkan kinerja algoritma Rabin-Karp.

Setelah dilakukan serangkaian uji coba algoritma Rabin-Karp yang dimodifikasi mempunyai waktu proses (*running time*) yang lebih baik dibandingkan algoritma Rabin-Karp tanpa modifikasi sedangkan untuk nilai *similarity* yang dihasilkan relatif sama.

Kata kunci: *text mining*, *string matching*, Rabin-Karp, plagiarisme dokumen, *stemming*, *hashing.*

SYSTEM DESIGN OF PLAGIARISM TEXT DOCUMENT DETECTION USING RABIN-KARP ALGORITHM

ABSTRACT

Plagiarism is copying someone's creation and admitting it as their own creation. Plagiarism is hard to be avoid. Therfore, many system are invented to detect document plagiarism, like MOSS, TESSY, JPlag, CopyCatch, etc.

The main idea to detect text plagiarism is by string matching. The algorithm used in this essay is Rabin-Karp algorithm. Rabin-Karp is superior in multiple pattern search. In this essay there will be some modification to improve Rabin-Karp performance.

After some experiments were done, it turn out that modified Rabin-Karp has better results in running time than the general Rabin-Karp algorithm, while the generated similarity values results of the two algorithm are not far different.

Key words: text mining, string matching, Rabin-Karp, plagiarsm, stemming, hashing.

KATA PENGANTAR

Alhamdulillahi rabbil 'alamin. Puji syukur penulis panjatkan kehadirat Allah SWT, atas segala rahmat dan karuniaNya, penulis dapat menyelesaikan skripsi yang berjudul "Perancangan Sistem Deteksi Plagiarisme Dokumen Teks dengan Menggunakan Algoritma Rabin-Karp".

Skripsi ini disusun dan diajukan sebagai syarat untuk memperoleh gelar sarjana pada program studi Ilmu Komputer, jurusan Matematika, fakultas MIPA, universitas Brawijaya.

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

1. Achmad Ridok, Drs., M.Kom selaku pembimbing I dan Bayu Rahayudi, ST., MT sebagai pembimbing II. Terima kasih atas semua waktu dan bimbingan yang telah diberikan.
2. Segenap bapak dan ibu dosen yang telah mendidik dan mengamalkan ilmunya kepada penulis.
3. Segenap staf dan karyawan di Jurusan Matematika FMIPA Universitas Brawijaya yang telah membantu kelancaran pengerjaan skripsi.
4. Papa, Mama, dan Adik. Terima kasih atas cinta, kasih sayang, doa, dukungan dan semangat yang tiada henti.
5. Verly Rahmadhani, Jayanti Utari, Mega Satya, Tresnaningtiyas, Widhy H, Dharma Surya, M. Chandra Saputra, atas bantuan, dukungan, semangat dan doanya.
6. Sahabat-sahabat ilkomers angkatan 2005, keluarga besar PPTI-UB dan seluruh warga Program Studi Ilmu Komputer Universitas Brawijaya.
7. Pihak lain yang telah membantu terselesaikannnya skripsi ini yang tidak bisa penulis sebutkan satu-persatu.

Penulis sadari bahwa masih banyak kekurangan dalam laporan ini disebabkan oleh keterbatasan kemampuan dan pengalaman. Oleh karena itu Penulis sangat menghargai saran dan

kritik yang sifatnya membangun demi perbaikan penulisan dan mutu isi skripsi ini untuk kelanjutan penelitian serupa di masa mendatang.

Penulis berharap semoga skripsi ini dapat memberikan manfaat kepada pembaca dan bisa diambil manfaatnya, baik oleh Penulis selaku mahasiswa maupun pihak-pihak lain yang tertarik untuk menekuni pengembangan *Text Mining*.

Malang, 11 Januari 2011

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

DAFTAR LAMPIRAN

BAB I
PENDAHULUAN

1.1. Latar Belakang

Semakin mudahnya pertukaran informasi dewasa ini tidak hanya membawa dampak positif bagi kemajuan teknologi, tetapi juga membawa dampak negatif yang hampir tidak dapat dihindari, yaitu plagiarisme.

Plagiarisme adalah suatu tindakan menjiplak karya seseorang dan kemudian mengakuinya sebagai karya sendiri. Tindakan plagiarisme ini sangatlah buruk tidak hanya bagi karya orang yang dijiplak tetapi juga orang yang melakukan tindakan plagiat ini. Plagiarisme dapat mematikan kreatifitas seseorang karena sudah terbiasa mengambil sesuatu yang bukan miliknya. Hal ini akan menimbulkan kecenderungan sikap malas dan tidak mau berfikir. Oleh karena itu, tindakan plagiarisme secara perlahan harus ditekan sejak dini.

Dengan memanfaatkan metode untuk pencocokan *string* pada dokumen, dapat dikembangkan untuk merancang aplikasi pendeteksi plagiarisme. Algoritma pencocokan *string* sendiri ada bermacam-macam, antara lain Boyer-Moore, *Brute Force*, Knuth-Morris-Pratt, Rabin-Karb, Smith-Waterman dan lain-lain.

Berdasarkan permasalahan di atas, sudah mulai dikembangkan aplikasi untuk mendeteksi plagiarisme seperti MOSS, JPlag, CopyCatch, EVE2 dan TESSY.

Pada jurnal sebelumnya (Firdaus, 2008) dilakukan penelitian secara skematis bagaimana cara kerja algoritma Rabin-Karp mendeteksi plagiarisme pada suatu dokumen, dalam makalah tersebut Firdaus menyatakan bahwa algoritma ini cocok untuk pencarian jamak (*multiple pattern*). Sedangkan pada penelitian yang dilakukan oleh R.Singh dan B. Kochar dengan judul *RB. Matcher: String Matching Technique* (Singh-Kochar, 2006) mengemukakan bagaimana meningkatkan performa dari algoritma Rabin-Karp sehingga dapat mengurangi waktu dalam proses pencocokan *string*.

Pada skripsi ini akan melakukan perancangan aplikasi dengan membandingkan kemiripan dokumen asli dengan dokumen yang ingin diuji. Dengan mengetahui persentase

kemiripan kedua dokumen tersebut dapat dijadikan bahan pertimbangan apakah dokumen yang diuji tersebut merupakan hasil menjiplak karya seseorang atau tidak.

Algoritma yang digunakan dalam skripsi ini adalah Rabin-Karb *algorithm*. Algoritma ini digunakan karena sangat efektif untuk pencarian lebih dari satu kata (*multi pattern*). (Karp-Rabin, 1987)

Di dalam algoritma Rabin-Karp menggunakan fungsi *hashing*. Fungsi *hashing* menyediakan metode sederhana untuk menghindari perbandingan jumlah karakter yang *quadratic* di dalam banyak kasus atau situasi. Daripada melakukan pemeriksaan terhadap setiap posisi dari teks ketika terjadi pencocokan pola, akan lebih efisien untuk melakukan pemeriksaan jika teks yang sedang diproses memiliki kemiripan seperti pada *pattern*. Untuk melakukan pengecekan kemiripan antara dua kata ini digunakan fungsi *hash*. (Fernando,2010)

Pada skripsi ini akan dilakukan beberapa modifikasi terhadap algoritma Rabin-Karp. Modifikasi yang dilakukan pada algoritma Rabin-Karp ini adalah dengan menyisipkan metode *Stemming* dengan menggunakan algoritma Arifin-Setiono pada tahap *preprorcessing*-nya dan melakukan modifikasi pada saat proses *hashing* serta perubahan pada proses *string-matching*. Kemudian akan dilakukan perbandingan Bagaimana perbedaan antara algoritma Rabin-Karp sebelum dan sesudah dimodifikasi dari sisi waktu proses dan keakuratan dalam mendeteksi kemiripan (*similarity*) dokumen.

1.2. Rumusan Masalah

Berdasarkan latar belakang diatas dapat diuraikan rumusan masalahnya, yaitu:

1. Bagaimana membuat sebuah sistem yang dapat melakukan pendeteksian plagiarisme terhadap dokumen teks?
2. Bagaimana perbandingan hasil nilai *similarity* dan waktu proses menggunakan algoritma Rabin-Karp sebelum dimodifikasi dengan algoritma Rabin-Karp yang telah dimodifikasi?

3. Bagaimana perbandingan nilai *similarity* dan waktu proses kedua algoritma dengan menggunakan *kgrams* yang berbeda?
4. Bagaimana persentase *error* yang dihasilkan oleh sistem terhadap nilai *similarity* yang dihasilkan?
5. Bagaimana pengaruh penggunaan *stemming* terhadap persentase similarity dan waktu proses pada algoritma Rabin-Karp?

1.3. Batasan Masalah

Batasan masalah dalam skirpsi ini, antara lain:

1. Hanya menguji data berupa teks, tidak menguji data berupa gambar maupun suara
2. Sistem tidak memperhatikan kesalahaan ejaan / penulisan pada dokumen.
3. Sistem tidak memperhatikan sinonim / persamaan kata
4. Data yang diuji bertipe .txt
5. Data yang diuji menggunakan bahasa Indonesia
6. *Kgram* yang digunakan 1 sampai dengan 5
7. Pada proses *stemming* mengabaikan imbuhan sisipan.

1.4. Tujuan Penelitian

Tujuan yang ingin dicapai dalam pembuatan skirpsi ini, antara lain:

1. Merancang aplikasi untuk mendeteksi plagiarisme dengan menggunakan algoritma Rabin-Karb sebelum dimodifikasi dan Rabin-Karp yang telah dimodifikasi
2. Mengetahui perbandingan persentase kemiripan (*similarity*) dan waktu proses antara dokumen asli dan dokumen yang di uji dengan menggunakan algoritma Rabin-Karp sebelum dimodifikasi dan algoritma Rabin-Karp yang telah dimodifikasi
3. Mengetahui perbandingan nilai *similarity* dan waktu proses kedua algoritma dengan menggunakan *kgrams* yang berbeda.
4. Mengetahui persentase *error* pada sistem terhadap nilai *similarity* yang dihasilkan dari pengujian.

5. Mengetahui pengaruh *stemming* terhadap persentase *similarity* dan waktu proses pada algoritma Rabin-Karp

1.5. Manfaat penelitian

Manfaat yang diharapkan dari pembuatan skripsi ini, antara lain:

1. Dapat membantu sebagai bahan pertimbangan dalam menentukan plagiarisme.
2. Dapat membandingkan hasil *similarity* dan waktu proses algoritma Rabin-Karp sebelum dimodifikasi dan algoritma Rabin-Karp yang telah dimodifikasi
3. Dapat menentukan persentase kemiripan (*similarity*) antara dokumen yang diuji dengan dokumen asli oleh sistem.

1.6. Metodologi Penelitian

1. Studi Literatur
   Mempelajari tentang sistem informasi *retrieval* dan metode pencocokan string melalui berbagai macam media, antara lain melalui internet, jurnal-jurnal dan buku yang berhubungan dengan *text processing.*

2. Perancangan Sistem
   Melakukan perancangan sistem dengan menguji algoritma yang digunakan terhadap data-data yang ada dan melakukan perhitungan manual apakah telah sesuai dengan yang diharapkan.

3. Implementasi
   Pembuatan aplikasi pendeteksi plagiarisme berdasarkan perancangan yang telah dibuat sebelumnya ke dalam program komputer.

4. Uji coba produk dan evaluasi.
   Melakukan uji coba program yang telah dibuat. Kemudian melakukan evaluasi terhadap kekurangan program dan memperbaikinya.

1.7. Sistematika Penulisan

Skripsi ini disusun dengan sistematika penulisan, sebagai berikut:

BAB I PENDAHULUAN

Pada bab ini dibahas mengenai latar belakang penulisan, rumusan masalah, batasan masalah, tujuan, manfaat dan sistematika penulisan skripsi ini.

BAB II TINJAUAN PUSTAKA

Pada bab ini dibahas mengenai pustaka yang digunakan dalam pengerjaan skripsi. Teori-teori yang terdapat pada bab ini mencakup *text processing* secara umum, metode pencocokan string, metode *hashing* dan sistem informasi *retrieval*.

BAB III PERANCANGAN DAN DESAIN SISTEM

Pada bab ini dibahas mengenai urutan langkah-langkah pengerjaan untuk mengidentifikasi plagiarisme, perancangan *user interface* dan disertai dengan perhitungan manual menggunakan algoritma Rabin-Karp

BAB IV IMPLEMENTASI DAN PEMBAHASAN

Pada bab ini dibahas tentang implementasi metode yang digunakan dalam hal ini adalah algoritma Rabin-Karp dalam mendeteksi plagiarisme dan uji coba terhadap program yang telah dibuat

BAB V KESIMPULAN DAN SARAN

Pada bab ini berisi tentang kesimpulan yang didapat dari pembuatan skripsi ini dan saran-saran yang mungkin dapat berguna dalam penelitian lebih lanjut.

BAB II
TINJAUAN PUSTAKA

2.1. Plagiarisme

2.1.1. Pengertian Plagiarisme

Plagiarisme adalah tindakan penyalahgunaan, pencurian / perampasan, penerbitan, pernyataan, atau menyatakan sebagai milik sendiri sebuah pikiran, ide, tulisan, atau ciptaan yang sebenarnya milik orang lain. (Ridhatillah, 2003)

Sedangkan menurut Kamus Besar Bahasa Indonesia (KBBI) Plagiarisme adalah penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri. (KBBI, 1997: 775)

Plagiat dapat dianggap sebagai tindak pidana karena mencuri hak cipta orang lain. Di dunia pendidikan, pelaku plagiarisme akan mendapat hukuman berat seperti dikeluarkan dari sekolah / universitas. Pelaku plagiat disebut sebagai plagiator.

Sistem pendeteksi plagiarisme dapat dikembangkan untuk :

1. Data teks seperti *essay*, artikel, jurnal, penelitian dan sebagainya.
2. Dokumen teks yang lebih terstruktur seperti bahasa pemrograman

Beberapa tipe plagiarisme yaitu :

1. *Word-for-word plagiarism* adalah menyalin setiap kata secara langsung tanpa diubah sedikitpun.
2. *Plagiarism of authorship* adalah mengakui hasil karya orang lain sebagai hasil karya sendiri dengan cara mencantumkan nama sendiri menggantikan nama pengarang yang sebenarnya.
3. *Plagiarism of ideas* adalah mengakui hasil pemikiran atau ide orang lain.

Plagiarism of sources, jika seorang penulis menggunakan kutipan dari penulis lainnya tanpa mencantumkan sumbernya. (Parvatti, 2005)

2.1.2. Metode Pendeteksi Plagiarisme

Metode pendeteksi plagiarisme dibagi menjadi tiga bagian yaitu metode perbandingan teks lengkap, metode dokumen *fingerprinting*, dan metode kesamaan kata kunci. Metode pendeteksi plagiarisme dapat dilihat pada gambar 2.1 : (Stein,2006)

Gambar 2.1 Metode pendeteksi plagiarisme

Berikut ini penjelasan dari masing-masing metode dan algoritma pendeteksi plagiarisme :

1. Perbandingan Teks Lengkap. Metode ini diterapkan dengan membandingkan semua isi dokumen. Dapat diterapkan untuk dokumen yang besar. Pendekatan ini membutuhkan waktu yang lama tetapi cukup efektif, karena kumpulan dokumen yang diperbandingkan adalah dokumen yang disimpan pada penyimpanan lokal. Metode perbandingan teks lengkap tidak dapat diterapkan untuk kumpulan dokumen yang tidak terdapat pada dokumen lokal. Algoritma yang digunakan pada metode ini adalah algoritma *Brute-Force*, algoritma *edit distance*, algoritma *Boyer Moore* dan algoritma *lavenshtein distance*
2. Dokumen *Fingerprinting*. Dokumen *fingerprinting* merupakan metode yang digunakan untuk mendeteksi keakuratan salinan antar dokumen, baik semua teks yang terdapat di dalam dokumen atau hanya sebagian teks saja. Prinsip kerja dari metode dokumen *fingerprinting* ini adalah dengan menggunakan teknik *hashing*. Teknik *hashing* adalah sebuah fungsi yang mengkonversi setiap

string menjadi bilangan. Misalnya Rabin-Karp, Winnowing dan Manber

3. Kesamaan Kata Kunci. Prinsip dari metode ini adalah mengekstrak kata kunci dari dokumen dan kemudian dibandingkan dengan kata kunci pada dokumen yang lain. Pendekatan yang digunakan pada metode ini adalah teknik dot.

2.2. Teks Mining

2.2.1. Pengertian Teks *Mining*

Teks *mining*, yang juga disebut sebagai Teks *Data Mining* (TDM) atau *Knowledge Discovery in Text* (KDT), secara umum mengacu pada proses ekstraksi informasi dari dokumen-dokumen teks tak terstruktur (*unstructured*). Teks *mining* dapat didefinisikan sebagai penemuan informasi baru dan tidak diketahui sebelumnya oleh komputer, dengan secara otomatis mengekstrak informasi dari sumber-sumber teks tak terstruktur yang berbeda. Kunci dari proses ini adalah menggabungkan informasi yang berhasil diekstraksi dari berbagai sumber (Tan,1999).

Tujuan utama teks *mining* adalah mendukung proses *knowledge discovery* pada koleksi dokumen yang besar.

Pada prinsipnya, teks *mining* adalah bidang ilmu multidisipliner, melibatkan *information retrieval* (IR), *text analysis*, *information extraction* (IE), *clustering*, *categorization*, *visualization*, *database technology*, *natural language processing* (NLP), *machine learning*, dan *data mining*. Dapat pula dikatakan bahwa teks *mining* merupakan salah satu bentuk aplikasi kecerdasan buatan *(artificial intelligence* / AI).

Teks *mining* mencoba memecahkan masalah *information overload* dengan menggunakan teknik-teknik dari bidang ilmu yang terkait. Teks *mining* dapat dipandang sebagai suatu perluasan dari *data mining* atau *knowledge-discovery in database* (KDD), yang mencoba untuk menemukan pola-pola menarik dari basis data berskala besar. Namun teks *mining* memiliki potensi komersil yang lebih tinggi dibandingkan dengan *data mining*, karena kebanyakan format alami dari penyimpanan informasi adalah berupa teks. Teks *mining* menggunakan informasi teks tak terstruktur dan mengujinya

dalam upaya mengungkap struktur dan arti yang "tersembunyi" di dalam teks.

Perbedaan mendasar antara teks *mining* dan *data mining* terletak pada sumber data yang digunakan. Pada *data mining*, pola-pola diekstrak dari basis data yang terstruktur, sedangkan di teks *mining*, pola-pola diekstrak dari data tekstual (*natural language*). Secara umum, basis data didesain untuk program dengan tujuan melakukan pemrosesan secara otomatis, sedangkan teks ditulis untuk dibaca langsung oleh manusia (Hearst, 2003).

2.2.2. Ruang Lingkup Teks *mining*

Teks *mining* merupakan suatu proses yang melibatkan beberapa area teknologi. Namun secara umum proses-proses pada teks *mining* mengadopsi proses *data mining*. Bahkan beberapa teknik dalam proses teks *mining* juga menggunakan teknik-teknik *data mining*. Ada empat tahap proses pokok dalam teks *mining*, yaitu pemrosesan awal terhadap teks (*text preprocessing*), transformasi teks (*text transformation*), pemilihan fitur (*feature selection*), dan penemuan pola (*pattern discovery*). (Even-Zohar, 2002).

a. *Text Preprocessing*

Tahap ini melakukan analisis semantik (kebenaran arti) dan sintaktik (kebenaran susunan) terhadap teks. Tujuan dari pemrosesan awal adalah untuk mempersiapkan teks menjadi data yang akan mengalami pengolahan lebih lanjut. Operasi yang dapat dilakukan pada tahap ini meliputi *part-of-speech* (PoS) *tagging*, menghasilkan *parse tree* untuk tiap-tiap kalimat, dan pembersihan teks.

b. *Text Transformation*

Transformasi teks atau pembentukan atribut mengacu pada proses untuk mendapatkan representasi dokumen yang diharapkan. Pendekatan representasi dokumen yang lazim digunakan adalah model "*bag of words*" dan model ruang vektor (*vector space model*). Transformasi teks sekaligus juga melakukan pengubahan kata-kata ke bentuk dasarnya dan pengurangan dimensi kata di dalam dokumen. Tindakan ini

diwujudkan dengan menerapkan *stemming* dan menghapus *stopwords*.

c. *Feature Selection*

Pemilihan fitur (kata) merupakan tahap lanjut dari pengurangan dimensi pada proses transformasi teks. Walaupun tahap sebelumnya sudah melakukan penghapusan kata-kata yang tidak deskriptif (*stopwords*), namun tidak semua kata-kata di dalam dokumen memiliki arti penting. Oleh karena itu, untuk mengurangi dimensi, pemilihan hanya dilakukan terhadap kata-kata yang relevan yang benar-benar merepresentasikan isi dari suatu dokumen. Ide dasar dari pemilihan fitur adalah menghapus kata-kata yang kemunculannya di suatu dokumen terlalu sedikit atau terlalu banyak.

Algoritma yang digunakan pada teks *mining*, biasanya tidak hanya melakukan perhitungan pada dokumen saja, tetapi juga pada *feature*. Empat macam *feature* yang sering digunakan:

1. *Character*, merupakan komponan individual, bisa huruf, angka, karakter spesial dan spasi, merupakan *block* pembangun pada level paling tinggi pembentuk semantik *feature*, seperti kata, *term* dan *concept*. Pada umumnya, representasi *character-based* ini jarang digunakan pada beberapa teknik pemrosesan teks.
2. *Words*.
3. *Terms* merupakan *single word* dan *multiword phrase* yang terpilih secara langsung dari *corpus*. Representasi *term-based* dari dokumen tersusun dari *subset term* dalam dokumen.
4. *Concept*, merupakan *feature* yang di-*generate* dari sebuah dokumen secara manual, *rule-based*, atau metodologi lain. (Triawati, 2009)

d. *Pattern Discovery*

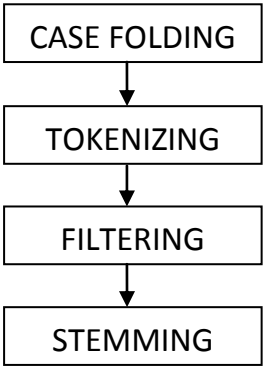
Pattern discovery merupakan tahap penting untuk menemukan pola atau pengetahuan (*knowledge*) dari keseluruhan teks. Tindakan yang lazim dilakukan pada tahap ini adalah operasi teks *mining*, dan biasanya menggunakan teknik-teknik

data mining. Dalam penemuan pola ini, proses teks *mining* dikombinasikan dengan proses-proses *data mining*.

Masukan awal dari proses teks *mining* adalah suatu data teks dan menghasilkan keluaran berupa pola sebagai hasil interpretasi atau evaluasi. Apabila hasil keluaran dari penemuan pola belum sesuai untuk aplikasi, dilanjutkan evaluasi dengan melakukan iterasi ke satu atau beberapa tahap sebelumnya. Sebaliknya, hasil interpretasi merupakan tahap akhir dari proses teks *mining* dan akan disajikan ke pengguna dalam bentuk visual. (Even-Zohar, 2002)

2.2.3. Ekstraksi Dokumen

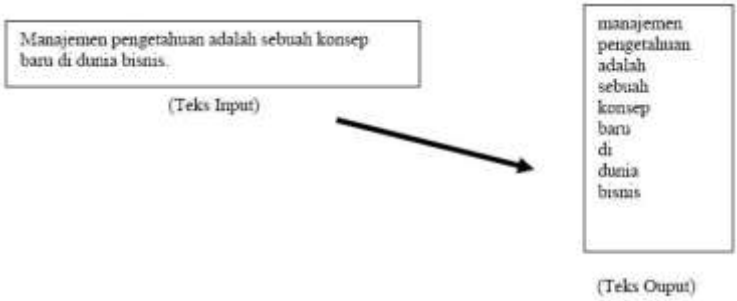
Teks yang akan dilakukan proses teks *mining*, pada umumnya memiliki beberapa karakteristik diantaranya adalah memiliki dimensi yang tinggi, terdapat *noise* pada data, dan terdapat struktur teks yang tidak baik. Cara yang digunakan dalam mempelajari suatu data teks, adalah dengan terlebih dahulu menentukan fitur-fitur yang mewakili setiap kata untuk setiap fitur yang ada pada dokumen. Sebelum menentukan fitur-fitur yang mewakili, diperlukan tahap *preprocessing* yang dilakukan secara umum dalam teks *mining* pada dokumen, yaitu *case folding, tokenizing, filtering, stemming, tagging* dan *analyzing*. Gambar 2.2 adalah tahap dari *preprocessing:* (Triawati, 2009)



Gambar 2.2 Tahap *preprocessing*

2.2.3.1 *Case folding* dan *Tokenizing*

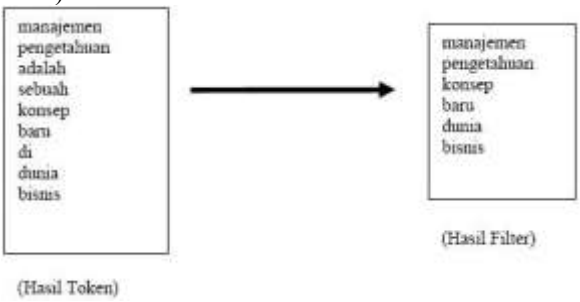
Case folding adalah mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf 'a' sampai dengan 'z' yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*. Tahap *tokenizing / parsing* adalah tahap pemotongan string input berdasarkan tiap kata yang menyusunnya. Gambar 2.3 adalah proses *tokenizing*: (Triawati, 2009)



Gambar 2.3 *Tokenizing*

2.2.3.2 *Filtering*

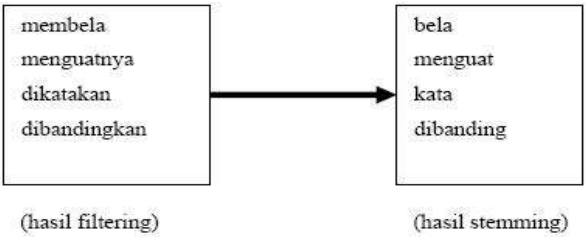
Filtering adalah tahap mengambil kata-kata penting dari hasil token. Bisa menggunakan algoritma *stoplist* (membuang kata yang kurang penting) atau *wordlist* (menyimpan kata penting). *Stoplist / stopword* adalah kata-kata yang tidak deskriptif yang dapat dibuang dalam pendekatan *bag-of-words*. Contoh *stopwords* adalah "yang", "dan", "di", "dari" dan seterusnya. Contoh dari tahapan ini dapat dilihat pada gambar 2.4: (Triawati, 2009)



Gambar 2.4 *Filtering*

2.2.3.3 *Stemming*

Tahap ***stemming*** adalah tahap mencari ***root*** kata dari tiap kata hasil ***filtering.*** Pada tahap ini dilakukan proses pengembalian berbagai bentukan kata ke dalam suatu representasi yang sama. Tahap ini kebanyakan dipakai untuk teks berbahasa nggris dan lebih sulit diterapkan pada teks berbahasa Indonesia. Hal ini dikarenakan bahasa Indonesia tidak memiliki rumus bentuk baku yang permanen. Contoh dari tahapan ini pada teks adalah seperti gambar 2.5: (Triawati, 2009)



Gambar 2.5 *Stemming*

2.3. Stemming

2.3.1. Algoritma *Stemming* Arifin

Algoritma ini didahului dengan pembacaan tiap kata dari file sampel. Sehingga input dari algoritma ini adalah sebuah kata yang kemudian dilakukan :

1. Pemeriksaan semua kemungkinan bentuk kata. Setiap kata diasumsikan memiliki 2 Awalan (prefiks) dan 3 Akhiran (sufiks).Sehingga bentuknya menjadi :

Prefiks 1 + Prefiks 2 + Kata dasar + Sufiks 3 + Sufiks 2 + Sufiks 1

Seandainya kata tersebut tidak memiliki imbuhan sebanyak imbuhan di atas, maka imbuhan yang kosong diberi tanda x untuk prefiks dan diberi tanda xx untuk sufiks.

2. Pemotongan dilakukan secara berurutan sebagai berikut :

AW : AW (Awalan)
AK : AK (Akhiran)
KD : KD (Kata Dasar)

a. AW I, hasilnya disimpan pada p1 (prefiks 1)
b. AW II, hasilnya disimpan pada p2 (prefiks 2)
c. AK I, hasilnya disimpan pada s1 (sufiks 1)
d. AK II, hasilnya disimpan pada s2 (sufiks 2)
e. AK III, hasilnya disimpan pada s3 (sufiks 3)

Pada setiap tahap pemotongan di atas diikuti dengan pemeriksaan di kamus apakah hasil pemotongan itu sudah berada dalam bentuk dasar. Kalau pemeriksaan ini berhasil maka proses dinyatakan selesai dan tidak perlu melanjutkan proses pemotongan imbuhan lainnya. Contoh pemenggalan kata "mempermainkannya":

a. Langkah 1 :
Cek apakah kata ada dalam kamus
Ya : Success
Tidak : lakukan pemotongan AW I
Kata = permainkannya
b. Langkah 2 :
Cek apakah kata ada dalam kamus
Ya : Success
Tidak : lakukan pemotongan AW II
Kata = mainkannya
c. Langkah 3 :
Cek apakah kata ada dalam kamus
Ya : Success
Tidak : lakukan pemotongan AK I
Kata = mainkan
d. Langkah 4 :
Cek apakah kata ada dalam kamus
Ya : Success
Tidak : lakukan pemotongan AK II
Kata = main
e. Langkah 5 :
Cek apakah kata ada dalam kamus
Ya : Success
Tidak : lakukan pemotongan AK III. Dalam
hal ini AK III tidak ada, sehingga kata tidak diubah.
Kata = main
f. Langkah 6

Cek apakah kata ada dalam kamus
Ya : Success
Tidak : "Kata tidak ditemukan"

3. Namun jika sampai pada pemotongan AK III, belum juga ditemukan di kamus, maka dilakukan proses kombinasi. KD yang dihasilkan dikombinasikan dengan imbuhan-imbuhannya dalam 12 konfigurasi berikut :

a. KD
b. KD + AK III
c. KD + AK III + AK II
d. KD + AK III + AK II + AK I
e. AW I + AW II + KD
f. AW I + AW II + KD + AK III
g. AW I + AW II + KD + AK III + AK II
h. AW I + AW II + KD + AK III + AK II + AK I
i. AW II + KD
j. AW II + KD + AK III
k. AW II + KD + AK III + AK II
l. AW II + KD + AK III + AK II + AK I

Sebenarnya kombinasi a, b, c, d, h, dan l sudah diperiksa pada tahap sebelumnya, karena kombinasi ini adalah hasil pemotongan bertahap tersebut. Dengan demikian, kombinasi yang masih perlu dilakukan tinggal 6 yakni pada kombinasi-kombinasi yang belum dilakukan (e, f, g, i, j, dan k). Tentunya bila hasil pemeriksaan suatu kombinasi adalah ‘ada’, maka pemeriksaan pada kombinasi lainnya sudah tidak diperlukan lagi.

Pemeriksaan 12 kombinasi ini diperlukan, karena adanya fenomena *overstemming* pada algoritma pemotongan imbuhan. Kelemahan ini berakibat pada pemotongan bagian kata yang sebenarnya adalah milik kata dasar itu sendiri yang kebetulan mirip dengan salah satu jenis imbuhan yang ada. Dengan 12 kombinasi itu, pemotongan yang sudah terlanjur tersebut dapat dikembalikan sesuai posisinya. (Arifin-Setiono, 2000)

2.4. *String Matching* (Pencocokan String/Kata)

2.4.1 Pengertian *String Matching*

String matching atau pencocokan *string* adalah suatu metode yang digunakan untuk menemukan suatu keakuratan/hasil dari satu atau beberapa pola teks yang diberikan. *String matching* merupakan pokok bahasan yang penting dalam ilmu komputer karena teks merupakan adalah bentuk utama dari pertukaran informasi antar manusia, misalnya pada literatur, karya ilmiah, halaman web dsb. (Hulbert-Helger,2007)

String matching digunakan dalam lingkup yang bermacam-macam, misalnya pada pencarian dokumen, pencocokan DNA *sequences* yang direpresentasikan dalam bentuk string dan juga *string matching* dapat dimanfaatkan untk mendeteksi adanya plagiarisme dalam karya seseorang.

String-matching fokus pada pencarian satu, atau lebih umum, semua kehadiran sebuah kata (lebih umum disebut *pattern*) dalam sebuah teks. Semua algoritma yang akan dibahas mengeluarkan semua kehadiran pola dalam teks. Pola dinotasikan sebagai x = x[0..m-1]; m adalah panjangnya.

Teks dinotasikan sebagai y = y[0..n-1]; n adalah panjangnya. Kedua string dibentuk dari set karakter yang disebut alphabet dinotasikan Σ dengan ukuran σ. (Atmopawiro, 2006)

2.5. Algoritma Rabin-Karp

Algoritma Karp-Rabin diciptakan oleh Michael O. Rabin dan Richard M. Karp pada tahun 1987 yang menggunakan fungsi *hashing* untuk menemukan *pattern* di dalam string teks. Karakteristik Algoritma Rabin-Karp : (Fernando, 2009)

- Menggunakan sebuah fungsi *hashing*
- Fase *prepocessing* menggunakan kompleksitas waktu O(m)
- Untuk fase pencarian kompleksitasnya : O(mn)
- Waktu yang diperlukan O(n+m)

Fungsi *hashing* menyediakan metode sederhana untuk menghindari perbandingan jumlah karakter yang quadratik di dalam banyak kasus atau situasi. Daripada melakukan pemeriksaan terhadap setiap posisi dari teks ketika terjadi pencocokan pola, akan lebih baik efisien untuk melakukan

pemeriksaan hanya jika teks yang sedang proses memiliki kemiripan seperti pada *pattern*. Untuk melakukan pengecekan kemiripan antara dua kata ini digunakan fungsi *hash*. (Fernando, 2009)

Untuk membantu algoritma ini, maka fungsi *hash* harus memenuhi hal-hal berikut ini :

- dapat dihitung dengan efisien
- memilik perbedaan yang tinggi untuk berbagai jenis *string*
- *hash* (y[j+1 .. j+m]) dapat dihitung dari *hash* (y[j.. j+m-1]) dan y[j+m]; yaitu : *hash* (y[j+1 .. j+m])= *rehash* (y[j], y[j+m], *hash* (y[j .. j+m-1]).

Untuk setiap *word* (8 bit) w yang memiliki panjang m, misalkan *hash* (w) didefinisikan sebagai berikut :

Hash (w[0 .. m-1])=(w[0]*2m-1+ w[1]*2m-2+···+ w[m-1]*20) mod q [2.1]

Dimana q merupakan bilangan prima yang cukup besar. Kemudian, lakukan *rehash* dengan rumus :

Rehash (a,b,h)= ((h-a*2m-1)*2+b) mod q [2.2]

Fase *prepocessing* dari algoritma Rabin-Karp mengandung perhitungan terhadap *hash* (x). Hal ini dapat dilakukan dalam waktu yang memilki kompleksitas O(m). Selama fase pencarian, hal yang perlu dilakukan cukup membandingkan *hash* (x) dengan

hash (y[j .. j+m-1]) untuk 0 <= j < n-m [2.3]

Jika kesamaannya ditemukan, masih perlu melakukan pemeriksaan kesamaan x=y[j..j+m-1] untuk karakter-karakter selanjutnya. Kompleksitas waktu untuk fase pencarian dari algoritma Rabin-Karp ini adalah O(mn). Diharapkan jumlah karakter teks yang dibandingkan adalah O(m+n). (Fernando, 2009)

Algoritma Rabin-Karp ini banyak digunakan dalam pendeteksian pencontekan atau kecurangan. Contohnya pada makalah atau pada *paper*. (Fernando, 2009)

Fungsi *hash* yang digunakan biasanya modulo berbasis bilangan prima besar. Alasan dipilih bilangan prima yang cukup

besar adalah untuk mengurangi kemungkinan dua buah *corresponding number value* yang sama. Sedangkan basis dipilih 10 karena banyaknya jenis karakter yang mungkin muncul adalah 0 sampai 9 berjumlah 10 jenis. (Andres, dkk, 2008)

2.5.1. *Hashing*

Hashing adalah suatu cara untuk mentransformasi sebuah string menjadi suatu nilai yang unik dengan panjang tertentu (*fixed-length*) yang berfungsi sebagai penanda string tersebut. Fungsi untuk menghasilkan nilai ini disebut fungsi *hash*, sedangkan nilai yang dihasilkan disebut nilai *hash*. Contoh sederhana *hashing* adalah:

Firdaus, Hari
Munir, Rinaldi
Rabin, Michael
Karp, Richard

menjadi :

7864 = Firdaus, Hari
9802 = Munir, Rinaldi
1990 = Rabin, Michael
8822 = Karp, Richard

Contoh di atas adalah pengunaan *hashing* dalam pencarian pada *database*. Apabila tidak di-*hash*, pencarian akan dilakukan karakter per karakter pada nama-nama yang panjangnya bervariasi dan ada 26 kemungkinan pada setiap karakter. Namun pencarian akan menjadi lebih efisien setelah di-*hash* karena hanya akan membandingkan empat digit angka dengan cuma 10 kemungkinan setiap angka. Nilai *hash* pada umumnya digambarkan sebagai *fingerprint* yaitu suatu string pendek yang terdiri atas huruf dan angka yang terlihat acak (data biner yang ditulis dalam heksadesimal). (Firdaus,2008)

Algoritma Rabin-Karp didasarkan pada fakta jika dua buah string sama maka harga *hash value*-nya pasti sama. Akan tetapi ada dua masalah yang timbul dari hal ini, masalah pertama yaitu ada

begitu banyak string yang berbeda, permasalahan ini dapat dipecahkan dengan meng-*assign* beberapa string dengan *hash value* yang sama. Masalah yang kedua belum tentu string yang mempunyai *hash value* yang sama cocok untuk mengatasinya maka untuk setiap string yang di-*assign* dilakukan pencocokan string secara *Brute-Force*. Kunci agar algoritma Rabin-Karp efisien, terdapat pada pemilihan *hash value*-nya. Salah satu cara yang terkenal dan efektif adalah memperlakukan setiap *substring* sebagai suatu bilangan dengan basis tertentu.(Firdaus, 2008)

2.5.2. *K-grams*

Kgrams adalah rangkaian *terms* dengan panjang K. Kebanyakan yang digunakan sebagai *terms* adalah kata. *K-gram* merupakan sebuah metode yang diaplikasikan untuk pembangkitan kata atau karakter. Metode *k-grams* ini digunakan untuk mengambil potongan-potongan karakter huruf sejumlah k dari sebuah kata yang secara kontinuitas dibaca dari teks sumber hingga akhir dari dokumen. Berikut ini adalah contoh *k-grams* dengan k=5 :

- *Text*: A do run run run, a do run run
- Kemudian dilakukanpenghilangan spasi : adorunrunrunadorunrun
- Sehingga dihasilkan rangkaian 5-*grams* yang diturunkan dari *text*:
- adoru dorun orunr runru unrun nrunr runru unrun nruna runad unado nador adoru dorun orunr runru unrun (Schleimer,dkk. 2003)

2.5.3. Konsep Algoritma Rabin-Karp

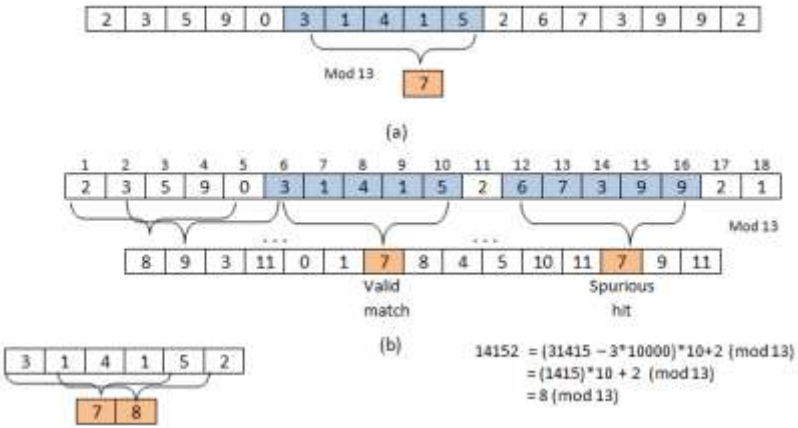
Algoritma Rabin-Karp adalah algoritma pencocokan *string* yang menggunakan fungsi *hash* sebgai pembanding antara *string* yang dicari (m) dengan *substring* pada teks (n). Apabila *hash value* keduanya sama maka akan dilakukan perbandingan sekali lagi terhadap karakter-karakternya. Apabila hasil keduanya tidak sama, maka substring akan bergeser ke kanan. Pergeseran dilakukan sebanyak (n-m) kali. Perhitungan nilai hash yang efisien pada saat pergeseran akan mempengaruhi performa dari algoritma ini. Cara

kerja dari algoritma Rabin-Karp dapat dilihat pada gambar 2.6 dan gambar 2.7. (Firdaus, 2008)

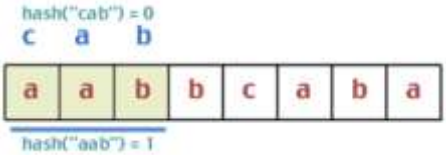
```
function RabinKarp (input s:
string[1..m], teks: string[1..n])
boolean
{  Melakukan pencarian string s pada
 string teks dengan algoritma Rabin-K}
Deklarasi
i : integer
ketemu = boolean

Algoritma:
ketemu ←false
hs ←hash(s[1..m])
for i ← 0 to n-m do
      hsub ←hash(teks[1..i+m-1])
      if hsub = hs then
         if teks[i..i+m-1] = s then
         ketemu ←true
      else
      hsub ←hash(teks[i+1..i+m])
endfor
return ketemu
```

Gambar 2.6 Algortima Rabin-Karp

Berikut ini adalah ilustrasi cara kerja algoritma Rabin-Karp: Diberikan masukan "cab" dan teks "aabbcaba". Fungsi *hash* yang dipakai misalnya akan menambahkan nilai keterurutan setiap huruf dalam alfabet (a = 1, b = 2, dst.) dan melakukan modulo dengan 3. Didapatkan nilai *hash* dari "cab" adalah 0 dan tiga karakter pertama pada teks yaitu "aab" adalah 1. Dapat dilihat pada gambar 2.7 :

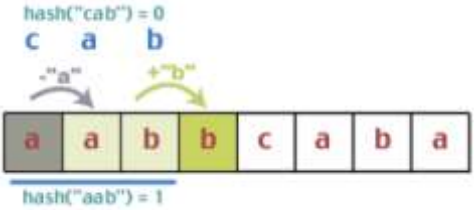


Gambar 2.7 Cara kerja algoritma Rabin-Karp

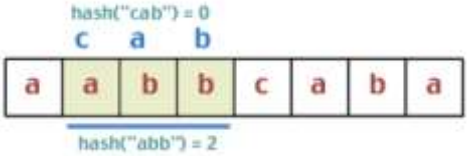


Gambar 2.8 Pengecekan tiga karakter pertama

Hasil perbandingan ternyata tidak sama, maka *substring* pada teks akan begeser satu karakter ke kanan. Algoritma tidak menghitung kembali nilai *hash substring*. Disinilah dilakukan apa yang disebut *rolling hash* yaitu mengurangi nilai karakter yang keluar dan menambahkan nilai karakter yang masuk sehingga didapatkan kompleksitas waktu yang relatif konstan pada setiap kali pergeseran. Setelah pergeseran, didapatkan nilai *hash* dari *fingerprint* "abb" (abb = aab – a + b) menjadi dua (2 = 1 – 1 + 2).. Proses pergeseran dan pengecekan dapat dilihat pada gambar 2.9 dan 2.10:

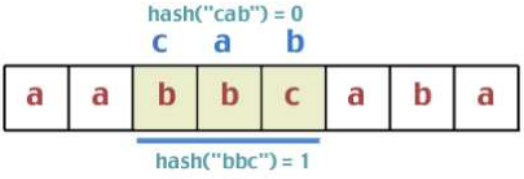


Gambar 2.9 Pengecekan terhadap *substring* berikutnya



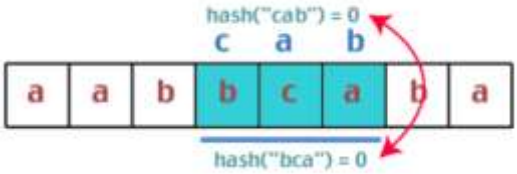
Gambar 2.10 Pengecekan *pattern* "c a b" dengan *substring* "a b b"

Hasil perbandingan juga tidak sama, maka dilakukan pergeseran. Begitu pula dengan perbandingan ketiga. Pada perbandingan keempat, didapatkan nilai *hash* yang sama. Gambar 2.11 pengecekan terhadap susbstring:



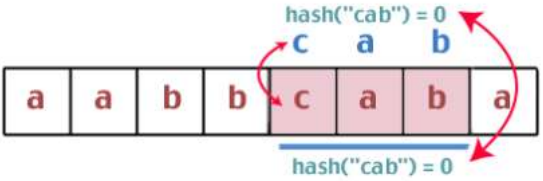
Gambar 2.11 Perbandingan *pattern* dengan *substring* berikutnya

Karena nilai *hash* sama, maka dilakukan perbandingan string karakter per karakter antara "bca" dan "cab". Didapatkan hasil bahwa kedua string tidak sama. Seperti pada gambar 2.12 :



Gambar 2.12 Perbandingan *pattern* yang mempunyai nilai *hash* sama dengan *substring*

Maka, kembali *substring* bergeser ke kanan. Pada perbandingan yang kelima, kedua nilai *hash* dan karakter pembentuk string sesuai, sehingga solusi ditemukan. Seperti pada gambar 2.13 :



Gambar 2.13 Hasil pencarian *pattern* ditemukan

Dari hasil perhitungan, kompleksitas waktu yang dibutuhkan adalah O(m+n) dengan m adalah panjang string masukan dan n adalah jumlah *looping* yang dilakukan untuk menemukan solusi. Hasil ini jauh lebih efisien daripada kompleksitas waktu yang didapat menggunakan algoritma *Brute-Force* yaitu O(mn). Algoritma Rabin-Karp ternyata masih kurang optimal dan cepat pada pencarian pola string tunggal (*single pattern search*) apabila dibandingkan dengan algoritma Boyer-Moore ataupun algoritma Knuth-Morris-Pratt, tetapi menjadi pilihan bila digunakan untuk mencari string dengan pola yang banyak (*multiple pattern search*). Bila pola yang ingin ditemukan memiliki panjang, sebut saja k, dan k bernilai besar (yang berarti string masukan panjang dan berpola banyak), algoritma Rabin- Karp dapat disesuaikan dengan tambahan penggunaan *filter* atau set data

structure, untuk mengecek apakah hasil *hashing* string masukan ada pada kumpulan nilai *hash* dari pola yang dicari.

Filter digunakan untuk mengeliminasi tanda baca (*punctuation*) dan beberapa kata dan kata sambung yang kurang signifikan untuk diberikan nilai *hash*, sedangkan set data *structure* adalah sekumpulan struktur data yang digunakan untuk membantu pencarian. Secara garis besar, *pseudocode* untuk algoritma Rabin-Karp untuk pencarian kumpulan *string* berpola banyak adalah: (diasumsikan semua string masukan pada himpunan s memiliki panjang yang sama dengan m)

Bila algoritma lain dapat mencari *string* berpola tunggal dalam waktu O(n), jika digunakan untuk mencari pola sebanyak k, maka akan membutuhkan waktu selama O(nk). Sedangkan varian Rabin-Karp di atas lebih efisien karena diharapkan dapat mencari dengan kompleksitas waktu O(n+k). (Firdaus, 2008)

2.5.4. *Multiple Pattern Search*

Algoritma Rabin-Karp apabila digunakan pada pencocokan *single pattern* masih kurang efisien dibandingkan dengan algoritma lain seperti KMP atau Boyer-Moore, karena kasus terburuk dari algoritma ini akan menghasilkan kompleksitas sebesar O(mn). Akan tetapi Rabin-Karp adalah sebuah algoritma yang tepat untuk pencocokan *multiple pattern*. Misal, ingin dicari bilangan yang besar (k), dapat dibuat varian sederhana dari algoritma Rabin-Karp yang memanfaatkan tabel *hash* atau struktur data lainnya untuk mengecek apakah string yang diperiksa termasuk himpunan *hash value* dari *pattern* yang dicari. *Pseudocode* dari pencocokan *multiple pattern* dengan menerapkan algoritma Rabin-Karp dapat dijelaskan pada gambar 2.14 : (Firdaus, 2008)

Di sini diasumsikan semua *substring* mempunyai panjang m, tetapi asumsi ini bisa dieliminir. Secara sederhana dengan membandingkan *hash value* sebelumnya dengan *hash value* dari semua *substring* secara secara berkelanjutan dengan melakukan pencarian di dalam himpunan data struktur, dan mengecek kecocokan dengan semua *substring* dengan *hash value* tersebut. Algoritma lainnya bisa memiliki kompleksitas O(n) untuk pencocokan *single pattern* dan kompleksitas O(nk) untuk pencocokan k *pattern*. Sebaliknya algoritma Rabin-karp diatas bisa

mencari k *pattern* dengan kompleksitas sebesar O(n+k). (Firdaus, 2008)

```
Function RabinkarpSetMultiplePattern (input teks: string [1..n], s: set of
string, m:integer) -> integer
Deklarasi
i : integer
str : string
ketemu = integer

Algoritma:
Ketemu ← 0
set hs ← (set kosong)
for each str in s do
Masukkan hash (s[1..m]) kedalam hs
    For i ← 0 to n-m
    Hsub ← hash (teks[i..i+m-1])
     if hsub = hs then
           if teks [i..i+m-1] = sebuah substring dengan hash hsub  then
           Ketemu ← ketemu+1

      else
           hsub <- hash (teks[i+1..i+m])
endfor
           Return ketemu
```

Gambar 2.14 Algoritma Rabin-Karp untuk *multiple pattern*

2.5.5. Pengukuran nilai *similarity*

Inti dari pendekatan *k-grams* dibagi menjadi dua tahap. Tahap pertama, membagi kata menjadi *k-grams*. Kedua, mengelompokkan hasil *terms* dari *k-grams* yang sama. Kemudian untuk menghitung *similarity* dari kumpulan kata tersebut maka digunakan *Dice's Similarity Coefficient* untuk pasangan kata yang digunakan. Nilai similaritas tersebut dapat dihitung dengan menggunakan :

[2.4]

Dimana S adalah nilai *similarity*, A dan B adalah jumlah dari kumpulan *K-grams* dalam teks 1 dan teks 2. C adalah jumlah *dari K-grams* yang sama dari teks yang dibandingkan. Berikut ini adalah contoh penghitungan nilai *similarity* 3 kata dengan K=2 (bigrams).

Kata yang dibandingkan (*)	*K-grams* yang sama	*Similarity*
Photography (9) dan *Photographic (10)*	Ph ho ot to gr ra ap = 8	2*8/(9+10) = 0.84
Photography (9) dan *Phonetic (7)*	Ph ho = 2	2*2/(9+7) = 0.25
Photographic (10) dan *Phonetic (7)*	Ph ho ic = 3	2*3/(10+7) = 0.35

* jumlah *k-grams* dari kata tersebut. (Kosinov, 2002)

2.5.6. Persentase nilai *similarity*

Untuk menentukan jenis plagiarisme antara dokumen yang diuji ada 5 jenis penilaian persentase *similarity:*

- 0% : Hasil uji 0% berarti kedua dokumen tersebut benar-benar berbeda baik dari segi isi dan kalimat secara keseluruhan
- < 15%: Hasil uji 15% berarti kedua dokumen tersebut hanya mempunyai sedikit kesamaan
- 15-50%: Hasil uji 15-50% berarti menandakan dokumen tersebut termasuk plagiat tingkat sedang
- >50%: Hasil uji lebih dari 50% berarti dapat dikatakan bahwa dokumen tersebut mendekati plagiarisme
- 100%: Hasil uji 100% menandakan bahwa dokumen tersebut adalah plagiat karena dari awal sampai akhir mempunyai isi yg sama persis. (Mutiara-Agustina, 2008)

2.5.7. Peningkatan Kinerja Algoritma Rabin-Karp

Seperti yang telah dijelaskan sebelumnya pada algoritma Rabin-Karp, *spurious hit* merupakan beban tambahan pada algoritma yang dapat menambah waktu proses karena harus membandingkan kembali tiap huruf dengan *pattern.* Hal ini terjadi ketika setelah dilakukan pencocokan ternyata *pattern* tersebut mempunyai nilai *hash* yang sama, tetapi setelah dilakukan pengecekan per karakter

ternyata *string* tersebut mempunyai urutan karakter yang berbeda sehingga membutuhkan waktu tambahan untuk melakukan pengecekan, jadi untuk menghindari pencocokan yang tidak perlu tersebut, dalam jurnal *RB_Matcher* mengatakan untuk pencocokan *string* harus memenuhi kriteria sebagai berikut:

$$REM(n1/q) = REM(n2/q) \quad [2.5]$$

$$QUOTIENT(n1/q) = QUOTIENT(n2/q) \quad [2.6]$$

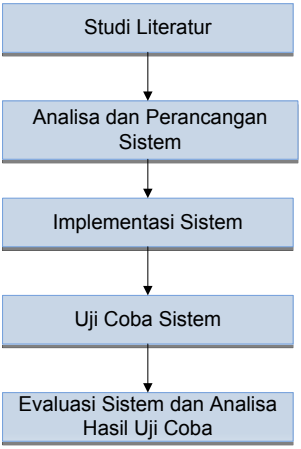
Jadi, untuk melakukan pencocokan *string* harus memenuhi 2 syarat diatas. Pertama *string* yang dicari dan *substring/pattern* pada teks harus mempunyai nilai *hash* modulo *(*setelah di-*modulo* dengan q*)* yang sama. Dan syarat yang kedua yaitu, hasil bagi (nilai *hash* dibagi dengan q) antara *string* yang dicari dan *substring* pada teks harus sama. Dengan demikian tidak diperlukan pengecekan kembali terhadap karakter-karakternya sehingga dapat membuat waktu proses lebih cepat dan efisien. (Singh-Kochar, 2008)

BAB III
PERANCANGAN DAN DESAIN SISTEM

Pada bab ini akan dibahas tentang perancangan sistem deteksi plagiarisme dengan menggunakan algoritma Rabin-Karb. Algoritma yang digunakan adalah algoritma Rabin-Karp biasa (berdasarkan jurnal Firdaus, 2008) dan algoritma Rabin-Karb yang telah dimodifikasi (Singh-Kochar,2008). Dalam perancangan sistem deteksi plasgiarisme ini, adapun langkah-langkah yang dilakukan adalah:

1. Mempelajari konsep algoritma Rabin-Karp yang digunakan dalam mendeteksi plagiarsme. Konsep yang dipelajari meliputi algoritma Rabin-Karb asli (yang belum dimodifikasi) dan algoritma Rabin-karp yang telah dimodifikasi. Dan mempelajari metode-metode penunjang lainnya yang akan digunakan dalam penelitian ini, seperti yang telah dijelaskan pada bab sebelumnya
2. Menganalisa dan merancang sistem untuk mendeteksi plagiarisme dengan menggabungkan beberapa metode yang telah dijelaskan sebelumnya.
3. Melakukan implementasi sistem berdasarkan analisa dan perancangan yang telah dilakukan sebelumnya.
4. Melakukan uji coba terhadap sistem yang telah dibuat dengan menganalisa hasil daripada sistem. Hasil yang dikeluarkan oleh sistem berupa waktu proses dan persentase kemiripan (*similarity*) antara dokumen teks yang diuji dengan dokumen teks asli dengan menggunakan algoritma Rabin-Karp asli (yang belum dimodifikasi) dan menggunakan algoritma Rabin-Karp yang telah dimodifikasi.
5. Mengevaluasi hasil kedua algoritma tersebut apakah perbedaan yang dihasilkan dari algoritma Rabin-Karp yang belum dimodifikasi dengan algoritma Rabin-Karp yang telah dimodifikasi.

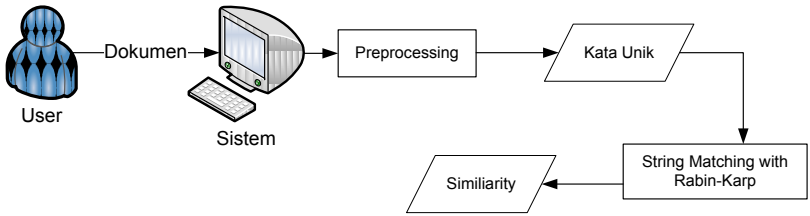
Langkah-langkah yang dilakukan dapat dilihat pada gambar 3.1 berikut:



Gambar 3.1 Diagram sistem

3.1. Perancangan Sistem Keseluruhan

Cara kerja sistem untuk deteksi plagiarisme ini adalah pertama kali memilih algoritma apa yang ingin digunakan, antara algoritma Rabin-Karp sebelum dimodifikasi atau algoritma Rabin-Karp yang telah dimodifikasi. Kemudian, user memasukkan dokumen teks yang ingin diuji dan dokument teks asli. Setelah itu sistem akan menganalisa persentase kemiripan (*similarity*) dan waktu prosesnya dengan menggunakan metode/algoritma yang telah dipilih sebelumnya. Gambar 3.2 adalah skema aliran data pada sistem.



Gambar 3.2 Skema aliran data

Data yang diuji dalam sistem ini adalah berupa dokumen teks. Dengan membandingkan hasil *similarity* dan waktu prosesnya dapat dianalisa efek dari modifikasi yang dilakukan terhadap algoritma Rabin-Karp tersebut apakah perbedaan dari kedua algoritma tersebut sebelum dimodifikasi dan setelah dilakukan modifikasi.

Perancangan aplikasi ini akan dibagi menjadi 2 bagian. Bagian yang pertama adalah aplikasi untuk mendeteksi plagiarisme dokumen dengan menggunakan algoritma Rabin-Karp sebelum dimodifikasi dan bagian yang kedua adalah algoritma Rabin-Karp yang telah dimodifikasi.

3.2. Perancangan Proses

Perancangan aplikasi yang dibuat adalah berupa sistem untuk mendeteksi plagiarisme suatu dokumen. Inputan pada aplikasi ini berupa dokumen teks yang mempunyai ekstensi .txt. User akan menginputkan 2 dokumen, yaitu dokumen asli dan dokumen yang ingin diuji. Setelah itu, sistem akan memproses kedua dokumen tersebut dan mengevaluasi berapakah *similarity* antara dokumen tersebut dan berapa lama waktu prosesnya.

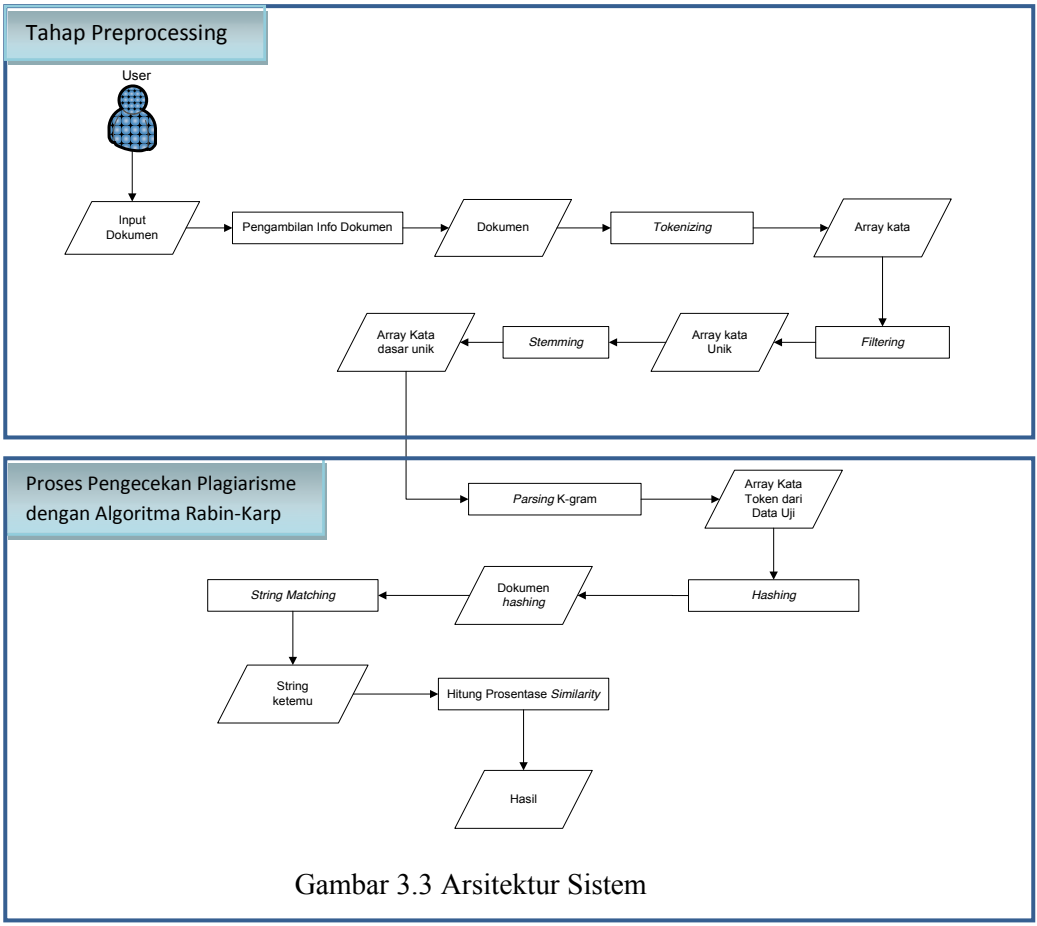
Pertama kali proses yang dilakukan oleh sistem adalah membaca *file* teks yang diinputkan oleh user. Dari dokumen yang telah diinputkan oleh user tadi, sistem akan melakukan pengecekan terhadap dokumen tersebut sehingga akan didapatkan informasi berupa jumlah kata, jumlah kalimat, jumlah paragraf dan ukuran dokumen tersebut.

Setelah sistem mendapatkan informasi dari dokumen yang telah diinputkan, sistem akan masuk ke tahap *preprocessing*. Pada tahap ini akan dilakukan beberapa proses, yaitu *tokenizing*, *filtering* (penghilangan kata yang tidak penting) dan *stemming* (pemotongan kata atau *term* menjadi kata dasar). Dapat dilihat pada gambar 3.5

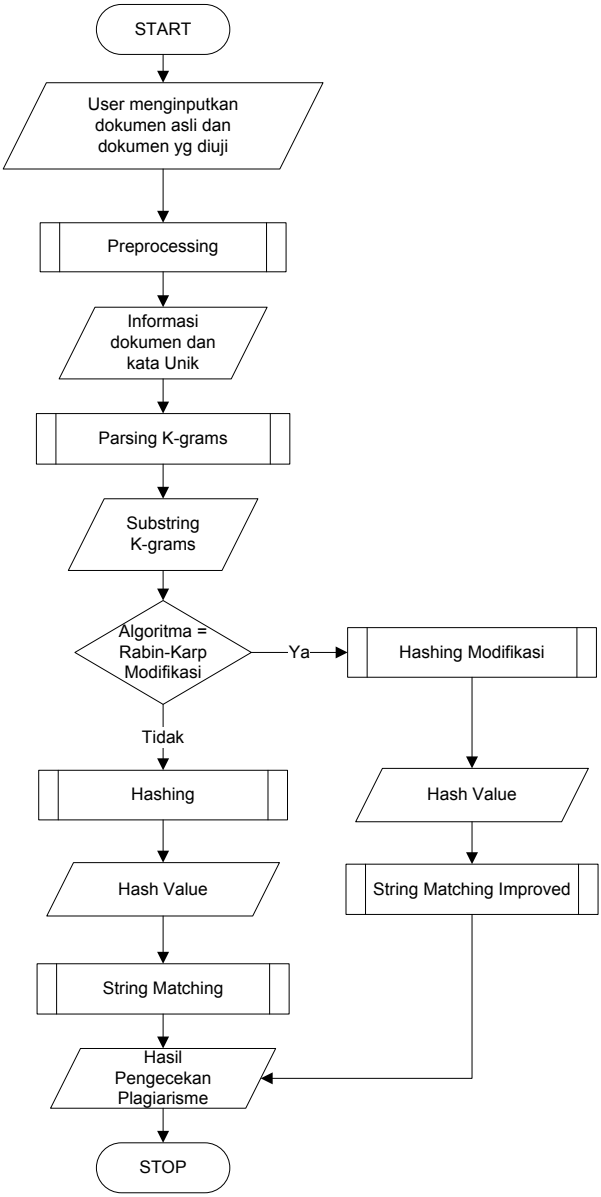
Proses *filtering* adalah proses penghilangan kata-kata dan tanda baca yang kurang penting, seperti kata "yang", "dan", "itu", spasi, koma dan sebagainya. Proses *Filtering* yang digunakan dalam sistem ini adalah menggunakan algoritma *stopword* dimana tiap kata (*term*) akan dicek apakah kata tersebut ada dalam daftar *stopword*. Jika terdapat dalam *stopword*, kata tersebut akan dihilangkan sehingga setelah dilakukan proses *filtering* akan didapatkan daftar kata unik.

Setelah proses *filtering* nantinya akan disisipkan proses *stemming*. Proses *stemming* adalah suatu proses pemotongan partikel-partikel seperti "-lah", "-kah", "-pun". Kemudian memotong kata ganti kepemilikan seperti "-ku" , "-mu", "-nya". Langkah berikutnya yaitu, pemotongan terhadap imbuhan sperti prefix (awalan) dan suffiks (akhiran) dan confix (awalan dan akhiran) pada kata unik seperti "di-", "-pun", "-kan" dan sebagainya, sehingga akan didpatkan kata dasarnya.

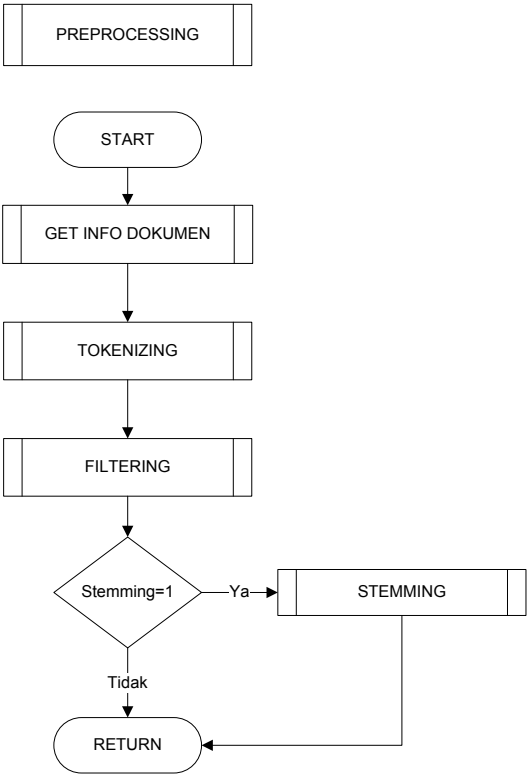
Gambar 3.3 adalah gambar dari proses pengecekan plagiarisme dokumen yang dilakukan oleh sistem



Gambar 3.3 Arsitektur Sistem



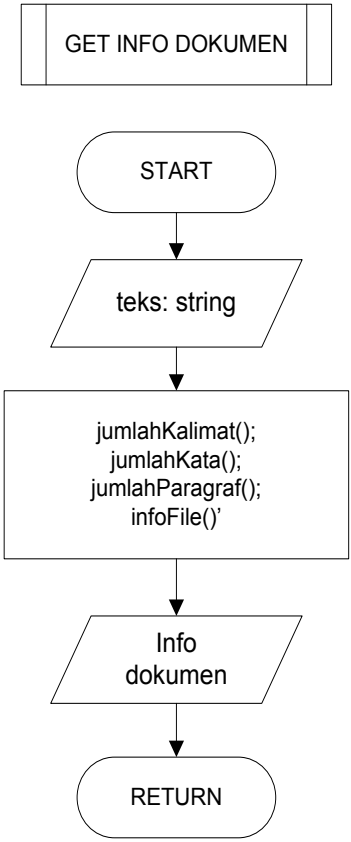
Gambar 3.4 *Flowchart* Proses Sistem



Gambar 3.5 *Flowchart* Preprocessing

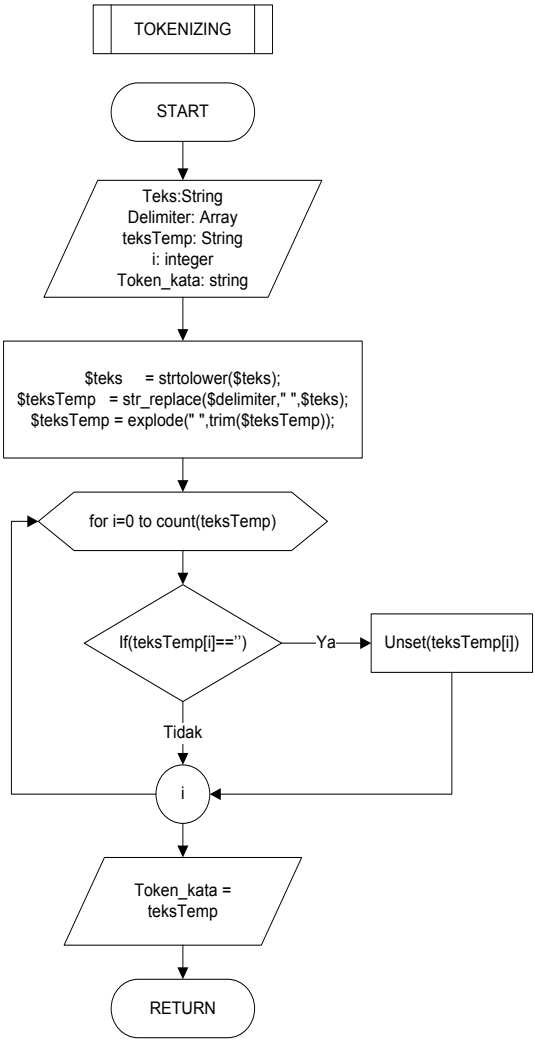
3.2.1. Preprocessing

Pada tahap *preprocessing* terdapat beberapa proses yang dilakukan oleh sistem terhadap dokumen yang diinputkan. Proses-proses tersebut adalah informasi dokumen, *tokenizing*, *case folding, filtering*, dan *stemming*. Proses mendapatkan informasi dokumen dapat dilihat pada gambar 3.6. Proses *tokenizing* adalah proses memecah kalimat menjadi potongan kata. Sedangkan proses *case folding* adalah proses merubah menjadi huruf kecil semua (*lowercase*) seperti pada gambar 3.7

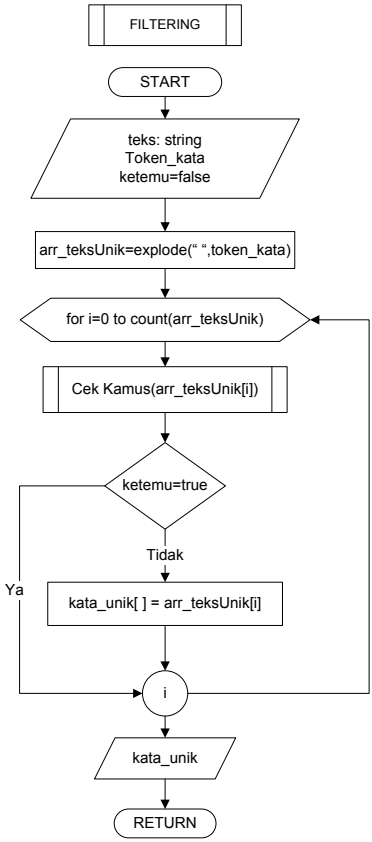


Gambar 3.6 *Flowchart* get info dokumen

Proses *filtering* adalah proses penghilangan partikel-partikel kata yang tidak penting sehingga didapatkan kata yang unik atau kata yang penting. Algoritma yang digunakan dalam proses *filtering* adalah algoritma *StopList*, yaitu menghilangkan kata yang terdapat didalam daftar kata yang telah dibuat sebelumnya. Gambar 3.7 adalah *Flowchart* dari proses *filtering.*

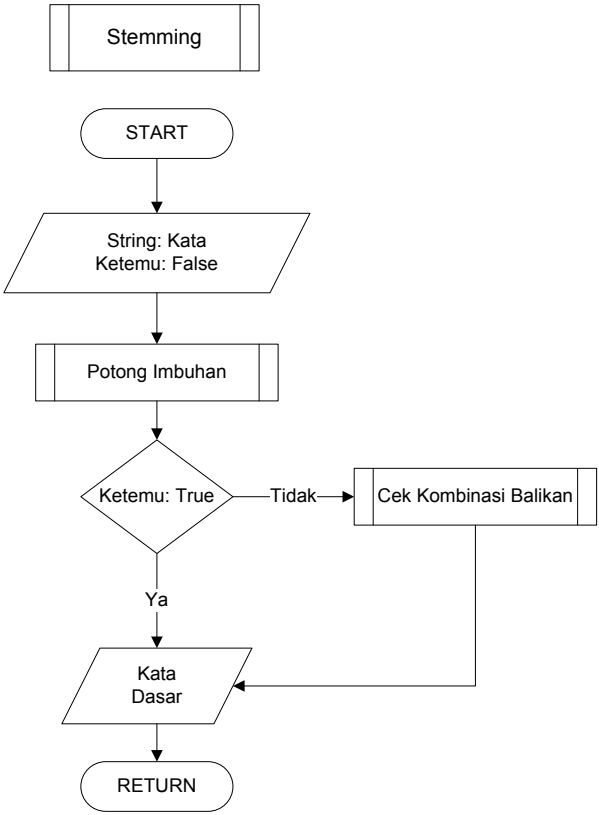


Gambar 3.7 *Flowchart* tokenizing



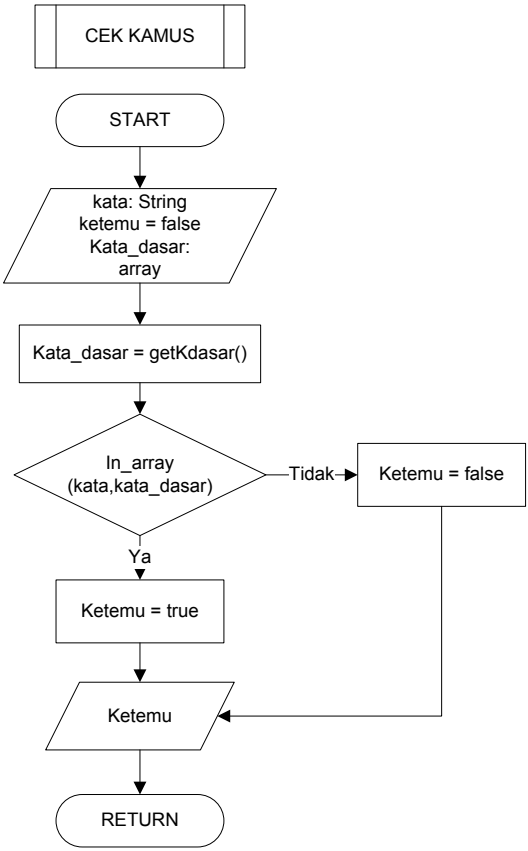
Gambar 3.8 *Flowchart* proses *filtering*

Proses *stemming* adalah proses pemotongan partikel-partikel *terms*/kata sehinnga menjadi kata dasar. Algoritma *stemming* yang diguanakan adalah *stemming* Arifin. Proses *stemming* ini digunakan untuk menangani masalah kata pasif-aktif dan perubahan partikel kata. *Stemming* yang digunakan adalah *stemming* arifin, seperti pada gambar 3.9. Untuk *flowchart stemming* Arifin selengkapnya dapat dilihat di lampiran



Gambar 3.9 *Flowchart Stemming* Arifin

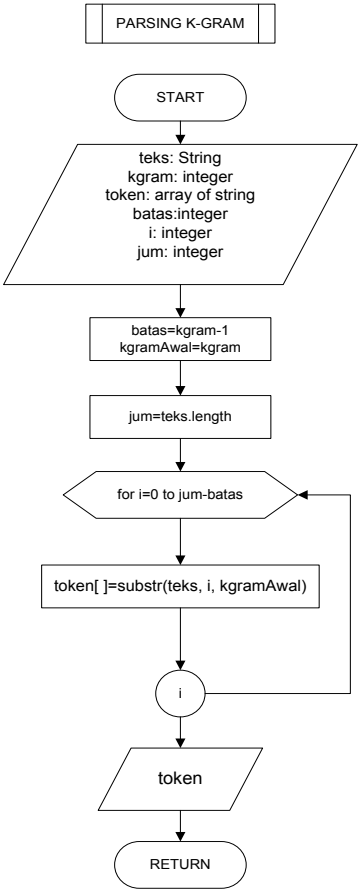
Sedangkan untuk proses cek kamus diambil data dari database kemudian disimpan dalam *array* kemudian dicocokkan antara kata yang dicari dengan kata yang terdapat didalam *array*. *Flowchart* cek kamus dapat dilihat pada gambar 3.10



Gambar 3.10 *Flowchart* cek kamus

3.2.2. Algoritma Rabin-Karp

Setelah melakukan *preprocessing* langkah selanjutnya adalah *tokenizing k-gram*, yaitu memecah kata menjadi potongan-potongan dimana setiap potongan mengandung karakter sebanyak *k*. Gambar 3.11 adalah proses *parsing k-grams*:



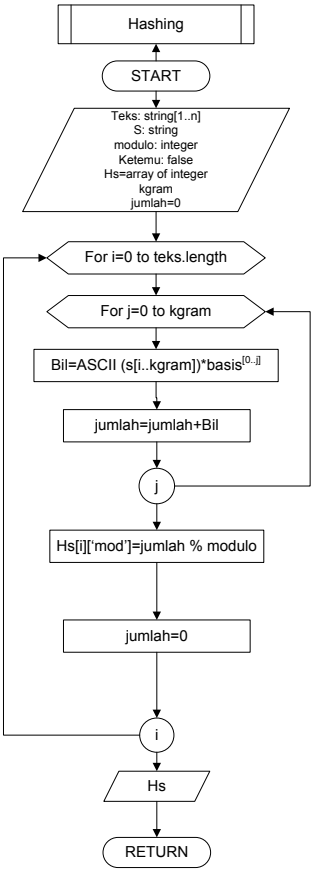
Gambar 3.11 *Flowcart* proses *parsing k-gram*

Setelah dilakukan *preprocessing* dan *parsing k-gram* langkah selanjutnya adalah implementasi dari algoritma Rabin-Karp.

Setelah tahapan *preprocessing* selesai, langkah berikutnya adalah implementasi dari algoritma Rabin-Karp. Langkah pertama yang dilakukan adalah menentukan *k-gram*. *K-gram* ini sendiri telah ditentukan sebelumnya pada saat proses *parsing k-gram*.

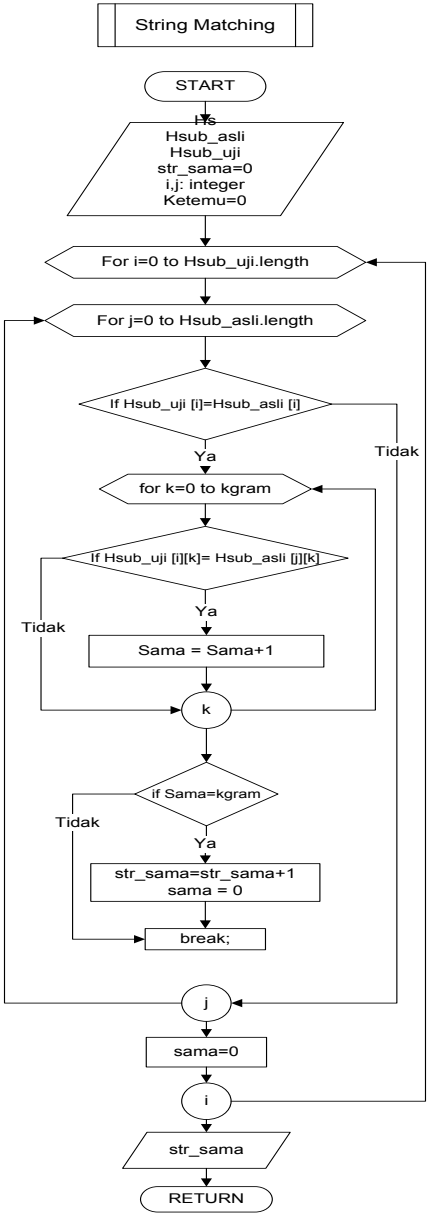
Langkah selanjutnya adalah melakukan proses *Hashing* terhadap seluruh pecahan string tadi yang telah dibagi menjadi k-

bagian pada set s. Gambar proses *Hashing* akan dijelaskan pada Gambar 3.12 :



Gambar 3.12 *Flowchart* Proses *Hashing* Rabin-Karp Sebelum dimodifikasi

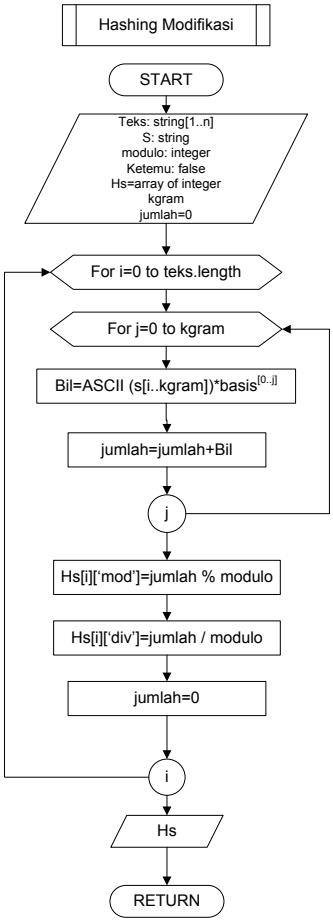
Setelah proses *Hashing* selesai maka, akan dilakukan pencocokan *string* dengan menggunakan algoritma Rabin-Karp. Gambar 3.13 dari proses *string-matching* pada algoritma Rabin-Karp



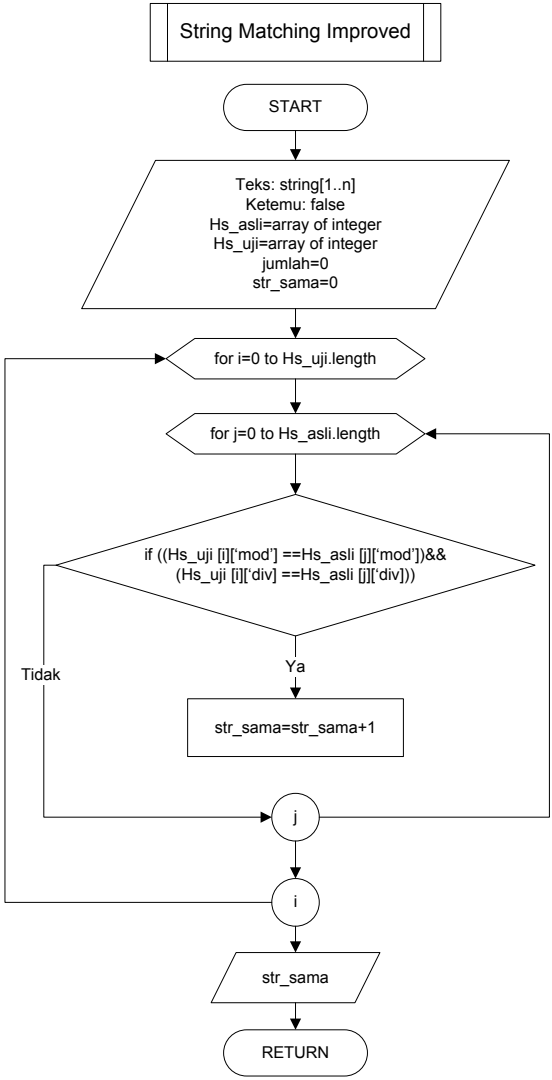
Gambar 3.13 *Flowchart* Algoritma *String-Matching* Rabin-Karp

3.2.3. Modifikasi Algoritma Rabin-Karp

Modifikasi yang dilakukan pada algoritma Rabin-Karp ini terletak pada penambahan proses ***stemming***, perubahan pada proses *Hashing* dan ***string-matching.*** Gambar 3.14 adalah *Flowchart* proses *Hashing* modifikasi. Sedangkan untuk *flowchart String Matching* yang telah dimodifikasi terdapat pada gambar 3.15.



Gambar 3.14 *Flowchart* proses *Hashing* modifikasi



Gambar 3.15 *Flowchart* proses *String Matching* modifikasi

3.3. Perancangan Uji Coba

3.3.1. Bahan Pengujian

Data yang diuji berupa dokumen teks yang mempunyai ekstensi .txt. Data diambil dari artikel pada blog-blog di internet dan dari buku studi literatur. Dokumen yang diuji akan dirubah sedemikian rupa untuk menguji apakah dokumen tersebut termasuk plagiat atau tidak dan bagaimana perbedaan dari algoritma Rabin-Karp sebelum dimodifikasi dan algoritma Rabin-Karp yang telah dimodifikasi.

3.3.2. Tujuan Pengujian

Tujuan dari pengujian sistem untuk medeteksi plagiarisme ini adalah sebagai berikut:

1. Menganalisa nilai *modulo* yang akan digunakan dalam proses *hashing* untuk kedua algoritma.
2. Menganalisa hasil *similarity* dan waktu proses yang dihasilkan oleh kedua algoritma berdasarkan nilai *kgram*.
3. Menganalisa prosentase error yang dihasilkan oleh algoritma Rabin-Karp sebelum dimodifikasi dan Rabin-Karp setelah dimodifikasi.
4. Membandingkan hasil deteksi plagiarisme dokumen dengan menggunakan algoritma Rabin-karp sebelum dimodifikasi dan algoritma Rabin-Karp yang telah dimodifikasi berdasarkan nilai *similarity* dan waktu prosesnya *(running time*).
5. Menganalisa pengaruh *stemming* terhadap nilai *similarity* dan waktu proses yang dihasilkan oleh kedua algoritma.

3.3.3. Perancangan Tabel Hasil Percobaan

Dengan membandingkan nilai *similarity* dan waktu proses antara algoritma Rabin-Karp sebelum dimodifikasi dan algoritma Rabin-Karp yang telah dimodifikasi, dapat ditentukan seberapa besar pengaruh modifikasi yang telah dilakukan terhadap algoritma Rabin-karp. Berikut ini adalah rancangan tabel perhitungan manual:

Tabel 3.1 Informasi dokumen

Kode	Ukuran File	Jumlah Kata	Jumlah Kalimat	Jumlah Paragraf

Tabel 3.2 *Hashing* data uji

No	Dok Asli	Dok Uji	Rabin-Karp		R-K Modif	
			t (s)	s (%)	t (s)	s (%)

Tabel 3.3 Uji Coba terhadap Sistem

No	Substring	Hash	Ketemu

3.3.4. Pengukuran nilai *Similarity*

Nilai *similarity* adalah kemiripan antara dokumen asli dan dokumen uji. Pengukuran nilai *similarity* yang dibandingkan adalah nilai *similarity* yang didapat dari hasil keluaran sistem dengan menggunakan algoritma Rabin-karp sebelum dimodifikasi dan algoritma Rabin-Karp yang telah dimodifikasi. Rumus yang digunakan telah dijelaskan pada bab 2.

3.3.5. Pengukuran waktu proses (*running time*)

Waktu proses disini adalah perbandingan seberapa lama waktu yang dibutuhkan oleh kedua algortima Rabin-Karb sebelum dimodifikasi dan Rabin-Karp yang telah dimodifikasi untuk melakukan seluruh proses dari awal hingga akhir sampai menghasilkan nilai *similarity*.

3.3.6. Pengukuran persentase *error*

Persentase *error* yaitu pengujian terhadap sistem dengan menghitung nilai *similarity* yang dihasilkan sistem dibandingkan nilai *similarity* yang diharapkan sehingga dapat dilihat apakah sistem yang dibuat telah sesuai dengan hasil yang diinginkan

3.3.7. Perancangan Dokumen Uji dan Dokumen Latih

Untuk dokumen latih yang digunakan pada skripsi ini ada beberapa jenis dokumen, ketentuan dari dokumen latih yang digunakan adalah sebagai berikut:

1. Sama: adalah dokumen latih yang isi teksnya sama dengan dokumen uji
2. 20% kata: adalah dokumen uji yang isi teksnya dilakukan pemotongan sebanyak 20% kata secara acak sehingga menghasilkan 80% kata yang sama.
3. 40% kata: adalah dokumen uji yang isi teksnya dilakukan pemotongan sebanyak 40% kata secara acak sehingga menghasilkan 60% kata yang sama
4. 60% kata: adalah dokumen uji yang isi teksnya dilakukan pemotongan sebanyak 60% kata secara acak sehingga menghasilkan 40% kata yang sama.
5. 80% kata: adalah dokumen uji yang isi teksnya dilakukan pemotongan sebanyak 80% kata secara acak sehingga menghasilkan 20% kata yang sama.
6. 20% tukar kalimat: adalah dokumen uji yang isi kalimatnya ditukar sebanyak 20% dari kalimat keseluruhan
7. 40% tukar kalimat: adalah dokumen uji yang isi kalimatnya ditukar sebanyak 40% dari kalimat keseluruhan.
8. 10% pasif-aktif: adalah dokumen uji yang 10% kata kerja didalamnya diganti menjadi pasif atau sebaliknya serta perubahan partikel penyusun katanya.

Jenis dokumen latih adalah dokumen uji yang telah dirubah sedemikian rupa untuk mengecek apakah sistem yang telah dibuat telah sesuai. Berikut ini adalah contoh data dokumen uji dan dokumen-dokumen latih:

a. Dokumen uji (A):

Plagiarisme adalah penjiplakan yang melanggar hak cipta. Pelaku plagiat disebut sebagai plagiator. Plagiator dapat dihukum berat.

b. Dokumen Latih A-00: *Dokumen latih tanpa dilakukan perubahan*

Plagiarisme adalah penjiplakan yang melanggar hak cipta. Pelaku plagiat disebut sebagai plagiator. Plagiator dapat dihukum berat.

c. Dokumen Latih A-20: *Dokumen latih dengan dilakukan pemotongan 20% kata*

Plagiarisme adalah penjiplakan yang melanggar hak. Pelaku disebut sebagai plagiator. Plagiator dapat dihukum.

d. Dokumen Latih A-40: *Dokumen latih dengan dilakukan pemotongan 40% kata*

adalah penjiplakan hak cipta. plagiat sebagai plagiator. Plagiator dihukum berat.

e. Dokumen Latih A-60: *Dokumen latih dengan dilakukan pemotongan 60% kata*

adalah penjiplakan hak cipta. Pelaku dapat dihukum.

f. Dokumen Latih A-80: *Dokumen latih dengan dilakukan pemotongan 80% kata*

hak Pelaku dapat dihukum.

g. Dokumen Latih A-20 Kal: *Dokumen latih yang ditukar 20% kalimatnya*

Plagiarisme adalah penjiplakan yang melanggar hak cipta. Pelaku plagiat disebut sebagai plagiator. Plagiator dapat dihukum berat.

h. Dokumen Latih A-40 Kal: *Dokumen latih yang ditukar 40% kalimatnya*

Plagiarisme adalah penjiplakan yang melanggar hak cipta. Plagiator dapat dihukum berat. Pelaku plagiat disebut sebagai plagiator.

i. Dokumen Latih A-10 Pasifaktif: *Dokumen latih dengan dilakukan perubahan 10% kata menjadi pasif atau sebaliknya*

Plagiarisme adalah menjiplak yang melanggar hak cipta. Pelaku plagiat disebut sebagai plagiator. Plagiator dapat menghukum berat.

3.3 Perancangan User Interface

3.3.1 Perancangan Input pada Sistem

Pada sistem deteksi plagiarisme dokumen ini ada beberapa inputan dan parameter yang harus diisi oleh user, yaitu:

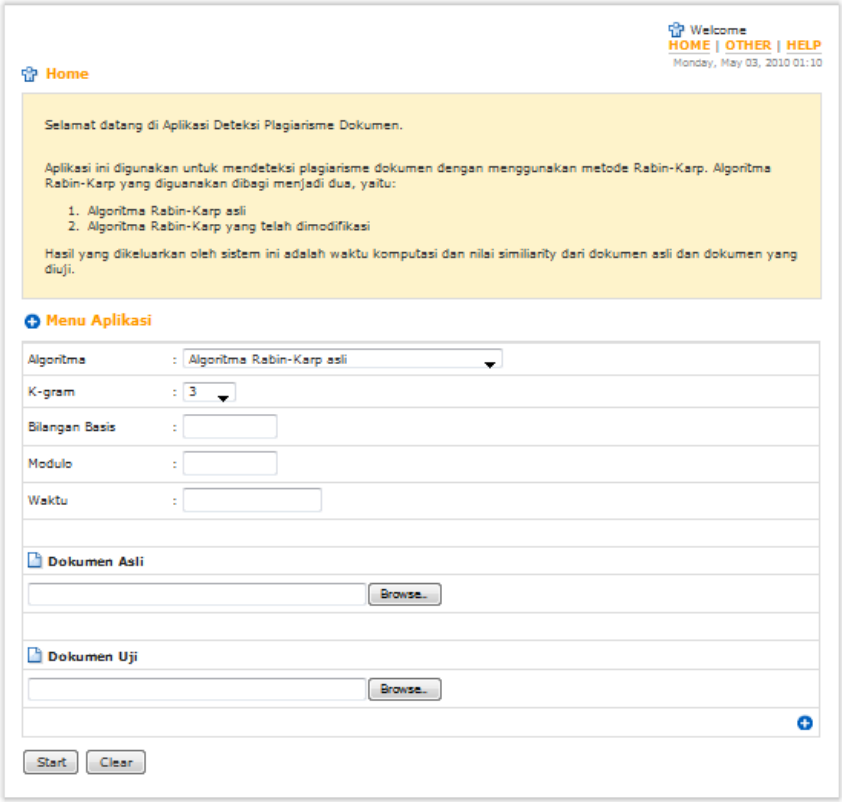
1. Sumber dokumen asli.
2. Sumber dokumen yang akan diuji.
3. Algoritma yang akan digunakan untuk proses deteksi plagiarisme dokumen.
4. Nilai *k-gram*.
5. Nilai *Modulo*.

3.3.2 Perancangan *Prototype* Sistem

Sistem untuk mendeteksi plagiarisme dokumen ini berbasis desktop dengan menggunakan bahasa PHP dan Javascript. Sistem ini akan membaca inputan yang diberikan oleh user, yaitu dokumen asli dan dokumen yang ingin diuji. Serta algoritma yang ingin digunakan dan penentuan *k-gram*. Setelah data yang diiputkan selesai, sistem akan memproses dokumen tersebut sehingga akan diperoleh informasi-informasi dari dokumen tersebut. Kemudian sistem akan masuk ke tahap *preprocessing* yang terdiri dari *filtering, stemming* dan *parsing k-gram*.

Gambar 3.19 adalah gambar rancangan *user interface* sistem. *Feature-feature* yang terdapat didalam sistem, antara lain:

1. Terdapat *field* Algoritma yang digunakan untuk memilih algoritma apa yang ingin digunakan
2. *Field k-gram* untuk menentukan *k-grams* yang akan digunakan
3. *Field* untuk memilih dokumen teks asli pada direktori komputer
4. *Field* untuk memilh dokumen yang ingin diuji pada direktori komputer



Gambar 3.19 prototype user interface sistem

3.4 Contoh Perhitungan Manual

Dok Uji: Plagiarisme adalah penjiplakan yang melanggar hak cipta. Pelaku plagiat disebut sebagai plagiator. Plagiator dapat dihukum berat.

Dok Latih A.00: Plagiarisme adalah penjiplakan yang melanggar hak cipta. Pelaku plagiat disebut sebagai plagiator. Plagiator dapat dihukum berat.

Hasil *tokenizing*, *filtering* dan *stemming*:

- Dokumen Uji:
  Plagiarismepenjiplakanmelanggarhakciptapelakuplagiatdisebutplagiatorplagiatordihukumberat
- Dokumen Latih:
  Plagiarismepenjiplakanmelanggarhakciptapelakuplagiatdisebutplagiatorplagiatordihukumberat

Hasil *Parsing K-gram* dan *Hashing*

No	Substring	Hashing
1	plag	
2	lagi	
3	agia	
4	giar	
5	iari	
6	aris	
7	rism	
8	isme	
9	smep	
10	mepe	
...	...	...
83	erat	

Pattern = "plag"
Hashing = $[(112*10^3)+(108*10^2)+(97*10^1)+(103*10^0)]$ mod 101
= (112000+10800+970+103) mod 101
= 123873/101
= 47

Pattern = "lagi"
Hashing = $[(108*10^3)+(97*10^2)+(103*10^1)+(105*10^0)]$ mod 101
= (10800+9700+1030+970+105) mod 101
= 118835/101
= 59

Pattern = "agia"
Hashing = $[(97*10^3)+(103*10^2)+(105*10^1)+(97*10^0)]$ mod 101
= (97000+10300+1050+97) mod 101

= 1108447/101
= 59

Pattern = "giar"
Hashing = $[(103*10^3)+(105*10^2)+(97*10^1)+(114*10^0)]$ mod 101
= (103000+10500+970+114) mod 101
= 114584/101
= 50

...

Perhitungan ini dilakukan pada semua hasil *parsing kgram* sehingga semua *substring* mempunyai nilai *hash*. Hal yang sama juga dilakukan pada dokumen latih kemudian nanti akan dicocokkan nilai hash dari setiap *substring* pada dokumen latih dengan dokumen uji. Kemudian dihitung jumlah *substring* yang ditemukan. Setelah itu akan dihitung nilai kemiripannya (*similarity*) dengan menggunakan rumus *Dice's Similarity Coeeficient*

Tabel 3.4 Informasi Dokumen Uji

No	Kode	Size (bytes)	Kalimat	Kata
1	A.00	129	3	16
2	A.20	110	3	13
3	A.40	82	3	10
4	A.60	51	2	7
5	A.80	25	1	4
6	AK.20	129	3	16
7	AK.40	129	3	16
8	AP.10	129	3	16

Data variabel untuk percobaan dokumen A adalah sebagai berikut:

- *Kgram* = 4
- *Modulo* = 101
- *Stemming* = tidak

Tabel 3.5 Percobaan dokumen A

No	Dok Asli	Dok Uji	Rabin-Karp		R-K Modif	
			t	s	t	s
1	A-asli.txt	A.00-kata.txt		100		100
2	A-asli.txt	A.20-kata.txt		77,5		77,5
3	A-asli.txt	A.40-kata.txt		66,18		66,18
4	A-asli.txt	A.60-kata.txt		40		40
5	A-asli.txt	A.80-kata.txt		14,15		14,15
6	A-asli.txt	AK.20-kal.txt		100		100
7	A-asli.txt	AK.40-kal.txt		88,27		88,27
8	A-asli.txt	AP.10-pa.txt		83,70		83,70

Keterangan:
t = waktu proses (detik)
s = *similarity* (%)

Tabel 3.6 Hasil pencocokan *string* terhadap dokumen uji

No	Substring	Hash	Ketemu
1	plag	47	√
2	lagi	59	√
3	agia	74	√
4	giar	50	√
5	iari	98	√
6	aris	81	√
7	rism	14	√
8	isme	26	√
9	smep	65	√
10	mepe	30	√
11	epen	99	√
12	penj	86	√
13	enji	45	√
14	njip	57	√
15	jipl	63	√
16	ipla	15	√
17	plak	51	√
18	laka	91	√
19	akan	3	√

20	kanm	42	√
...	...	...	...
86	erat	63	√

Similarity = 100%

Untuk data nilai hash, parsing *kgram* dan hasil pengujian terhadap dokumen A, selengkapnya dapat dilihat pada lampiran.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

Dalam tahap implementasi sistem ada beberapa syarat yang harus disiapkan sebelumnya. Syarat-syarat tersebut meliputi perangkat keras (*hardware*) dan perangkat lunak (*software*).

4.1. Lingkungan Implementasi

Lingkungan implementasi meliputi lingkungan perangkat keras dan perangkat lunak

4.1.1. Lingkungan Perangkat Keras

Dalam perancangan dan pengembangan sistem deteksi anti-plagiarisme ini menggunakan komputer (PC) dengan spesifikasi:

1. Prosesor Intel Core2Duo E2140 @1.60GHz
2. VGA 256 ATI Radeon X1050
3. Memory 1 GB
4. Harddisk 320 GB
5. Monitor 17”
6. Motherboard dan keyboard

4.1.2. Lingkungan Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem deteksi anti-plagiarisme ini adalah:

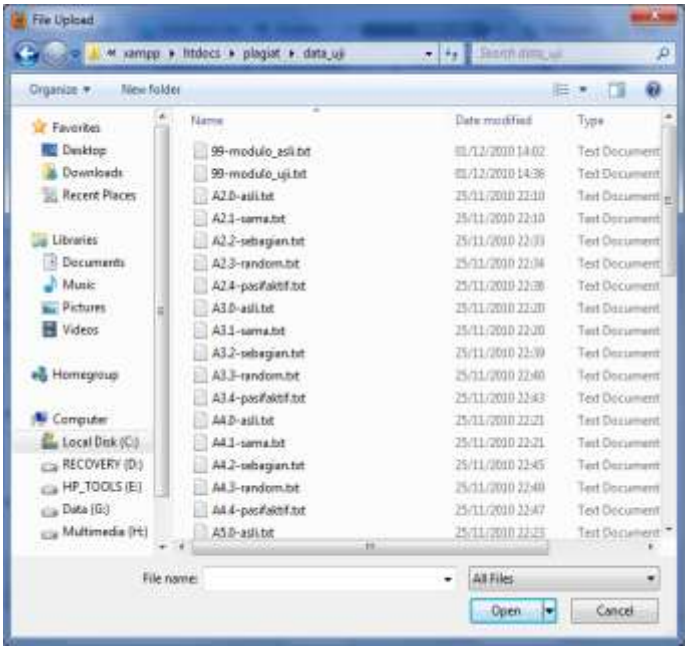
1. Sistem Operasi Windows XP SP2
2. XAMPP 1.6.3
3. PHP Version 5.2.3
4. Apache 2.0
5. MySQL 5.0.45
6. Heidi SQL 4.0
7. PHP Designer 7.0
8. Mozilla Firefox 3.6

4.2. Implementasi User Interface

Berdasarkan perancangan *user interface* yang telah dilakukan pada bab 3, maka dihasilkan user interface seperti pada gambar 4.1. Pada halaman utama terdapat beberapa *field* yang harus diisi, yaitu

field algoritma untuk memilih algoritma yang hendak digunakan dalam proses pencocokan string. *Field stemming* akan ditambahkan pada saat *preporcessing* jika dipilih "ya". Field *modulo* untuk memilih *modulo* yang akan digunakan dalam proses *hashing*. *Field* dokumen asli dan dokumen uji merupakan *field* untuk melakukan *upload file* yang akan diuji seperti pada gambar 4.2.

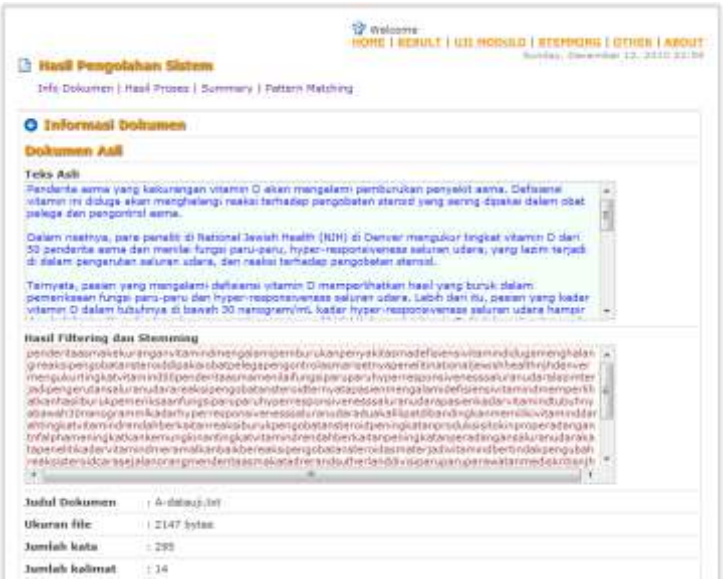
Gambar 4.1 Halaman utama



Gambar 4.2 *Upload* file teks

Setelah semua *field* diisi maka akan dilakukan proses pengecekan plagiarisme dokumen. Pada halaman Hasil Pengolahan Sistem terdapat kalimat unik hasil dari proses *tokenizing*, *filtering* dan *stemming* yang telah dilakukan sistem, *parsing kgram*, nilai *hash* , jumlah *substring* yang ditemukan (*string matching*) dan hasil dari proses yang berupa nilai *similarity* dan waktu proses (*running time*). Contoh Halaman Pengolahan Sistem dapat dilihat pada gambar 4.3. dan 4.4.

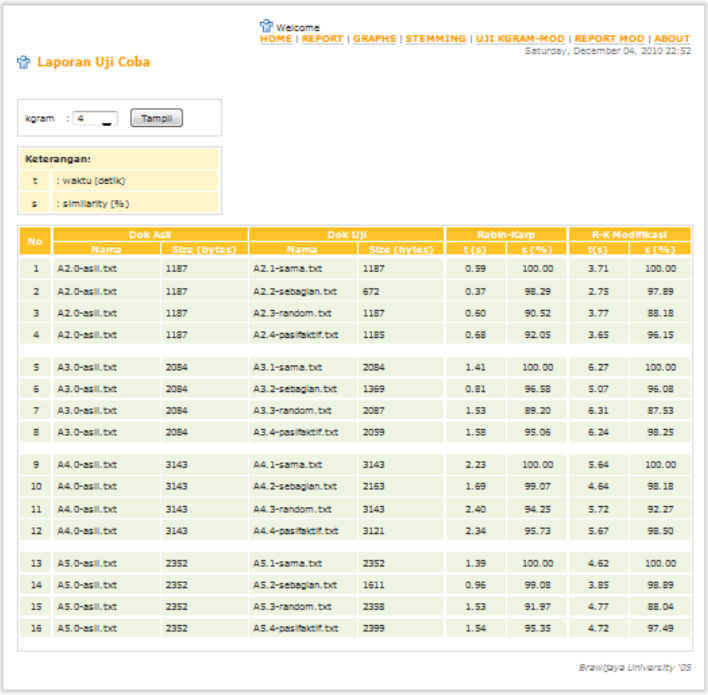
Pada gambar 4.3 memperlihatkan hasil *tokenizing*, *filtering* dan *stemming*. Sedangkan gambar 4.4 memperlihatkan rangkuman hasil proses pengecekan plagiarisme.



Gambar 4.3 Hasil proses *tokenizing*, *filtering* dan *stemming*

Gambar 4.4 *Summary* hasil proses pengecekan plagiarisme

Setelah diperoleh nilai *similarity* dan waktu prosesnya, hasil tersebut dapat disimpan dalam *database*. Kemudian rangkuman hasil tersebut dapat digunakan untuk laporan uji coba yang dapat dilihat pada gambar 4.5. laporan hasil uji coba tersebut dibagi berdasarkan *kgram* dan *stemming.*



Welcome
HOME | REPORT | GRAPHS | STEMMING | UJI KGRAM-MOD | REPORT MOD | ABOUT
Saturday, December 04, 2010 22:52

Laporan Uji Coba

kgram : 4 Tampil

Keterangan:
t : waktu (detik)
s : similarity (%)

No	Dok Asli		Dok Uji		Rabin-Karp		R-K Modifikasi	
	Nama	Size (bytes)	Nama	Size (bytes)	t (s)	s (%)	t(s)	s (%)
1	A2.0-asli.txt	1187	A2.1-sama.txt	1187	0.59	100.00	3.71	100.00
2	A2.0-asli.txt	1187	A2.2-sebagian.txt	672	0.37	98.29	2.75	97.89
3	A2.0-asli.txt	1187	A2.3-random.txt	1187	0.60	90.52	3.77	88.18
4	A2.0-asli.txt	1187	A2.4-pasifaktif.txt	1185	0.68	92.05	3.65	96.15
5	A3.0-asli.txt	2084	A3.1-sama.txt	2084	1.41	100.00	6.27	100.00
6	A3.0-asli.txt	2084	A3.2-sebagian.txt	1369	0.81	96.58	5.07	96.08
7	A3.0-asli.txt	2084	A3.3-random.txt	2087	1.53	89.20	6.31	87.53
8	A3.0-asli.txt	2084	A3.4-pasifaktif.txt	2059	1.58	95.06	6.24	98.25
9	A4.0-asli.txt	3143	A4.1-sama.txt	3143	2.23	100.00	5.64	100.00
10	A4.0-asli.txt	3143	A4.2-sebagian.txt	2163	1.69	99.07	4.64	98.18
11	A4.0-asli.txt	3143	A4.3-random.txt	3143	2.40	94.25	5.72	92.27
12	A4.0-asli.txt	3143	A4.4-pasifaktif.txt	3121	2.34	95.73	5.67	98.50
13	A5.0-asli.txt	2352	A5.1-sama.txt	2352	1.39	100.00	4.62	100.00
14	A5.0-asli.txt	2352	A5.2-sebagian.txt	1611	0.96	99.08	3.85	98.89
15	A5.0-asli.txt	2352	A5.3-random.txt	2358	1.53	91.97	4.77	88.04
16	A5.0-asli.txt	2352	A5.4-pasifaktif.txt	2399	1.54	95.35	4.72	97.49

Brawijaya University '05

Gambar 4.5 Laporan hasil uji coba dengan *kgram* yang berbeda

4.3. Implementasi Program

4.3.1. Kelas dan Fungsi

Dalam melakukan tahap implementasi sistem, dibentuk struktur data yang terdiri dari kelas-kelas utama yang didalamnya memiliki fungsi-fungsi yang menunjang dalam pembuatan sistem. Kelas-kelas tersebut terdiri dari kelas File, kelas Rabin-Karp, dan kelas Rabin-Karp modifikasi

1. Kelas File

Kelas File ini digunakan untuk memperoleh informasi dokumen yang nantinya akan digunakan untuk mengolah *file* pada proses inti. Pada kelas File ini terdapat beberapa fungsi, yaitu konstruktor kelas File yang berisi deklarasi *path file* yang akan diuji, fungsi setFilename merupakan fungsi yang digunakan untuk menge-set nama *file*, fungsi setPath digunakan untuk menentukan *path* dari *file* yang akan diuji, getPath digunakan untuk mendapatkan *path* dari file, getFullpath digunakan untuk mendapatkan *path* beserta nama filenya, yang nantinya akan digunakan untuk membaca *file*.

Fungsi bacaFile digunakan untuk membaca *file* txt pada direktori. Fungsi jumlahKalimat, jumlahParagraf dan jumlahKata digunakan untuk menghitung jumlah kalimat, pararagraf dan kata pada dokumen. Fungsi fileInfo() yang digunakan untuk mendapatkan informasi ukuran dari dokumen yang diuji.

Serta ada beberapa fungsi lainnya yang digunakan untuk keperluan laporan uji coba dan penyimpanan hasil uji pada database, yaitu fungsi insertDB digunakan untuk menyimpan hasil pettern matching dari algoritma Rabin-Karp. Fungsi insertUjiMod merupakan fungsi yang digunakan untuk menyimpan hasil uji coba modulo terhadap similarity dan waktu proses. Fungsi laporanUjicoba digunakan untuk menampilkan hasil uji coba yang telah disimpan dalam database. Fungsi laporan UjiMod digunakan untuk menampilkan laporan hasil uji modulo yang telah disimpan di database dan fungsi bilangan_prima digunakan untuk menampilkan bilangan prima mulai dari 0 sampai n.

2. Kelas Rabin-Karp

Kelas ini adalah implementasi dari algoritma Rabin-Karp tanpa dilakukan modifkasi. Fungsi-fungsi yang terdapat pada kelas ini antara lain, fungsi setVar() digunakan untuk mengeset variabel global untuk basis, *kgram* dan modulo. Sebelum masuk ke proses inti terdapat tahap *preprocessing* yang terdiri dari beberapa fungsi, yaitu fungsi tokenizingSubstring() yang digunakan untuk mengubah kalimat-kalimat pada teks menjadi huruf kecil semua (*casefolding*) dan menghilangkan tanda baca.

Fungsi filtering() digunakan untuk menghilangkan kata-kata tidak penting (*stop-word*) sehinga hasil dari filtering ini akan

didapatkan sekumpulan kata-kata penting saja. Data *stop-word* yang digunakan diambil dari jurnal Tala. Setelah dilakukan tahapan *preprocessing* kemudian masuk ke tahap inti dari algoritma ini yang terdiri dari beberapa fungsi, yaitu fungsi *kgram*Parsing() yang digunakan untuk memecah teks menjadi substring-substring tergantung dari *kgram* yang dipilih nantinya.

Fungsi *hashing* digunakan untuk mengubah *substring* yang telah di-*parsing* menjadi bilangan-bilangan dengan menggunakan rumus *hash* yang merupakan ciri khas dari algoritma ini. Fungsi *string matching* digunakan untuk mencari/mencocokkan substring pada dokumen uji dan dokumen asli. Kemudian fungsi similarityNgram() digunakan untuk menghitung persentase kemiripan antara dokumen uji dan dokumen asli.

3. Kelas Rabin-Karp Modifikasi

Pada intinya kelas Rabin-Karp Modifikasi hampir sama dengan kelas Rabin-Karp tetapi ada sedikit perubahan dan penambahan pada fungsi *hashing*, *string maching* dan *stemming.*

Fungsi getKdasar digunakan untuk mengambil semua kata dasar yang terdapat pada *database*. Kata dasar diambil dari KBBI online. Fungsi *stemming* yang digunakan berdasarkan algoritma yang telah dikembangkan oleh Arifin dan Setiono. Fungsi *stemming* ini akan ditambahkan pada saat tahap *preprocessing*.

4.3.2. Tahap *Preprocessing*

Pada tahap *preprocessing* akan dilakukan pengumpulan informasi dokumen yaitu bacaFile() yang terdapat pada gambar 4.6. penghitungan jumlah kata, kalimat, dan paragraf dari dokumen teks yg diinputkan user dengan menggunakan fungsi jumlahKata(), jumlahKalimat() dan jumlahParagraf yang terdapat pada gambar 4.7, sedangkan untuk mendapatkan ukuran (*size*) dokumen dengan menggunakan fungsi infoFile() dapat dilihat pada gambar 4.8

Tahap *preporcessing* terdiri dari *tokenizing*, *filtering* dan *stemming*. Inputan pada tahap *tokenizing* adalah dokumen teks yang diinputkan oleh user. *Output* dari proses *tokenizing* akan menghasilkan *array* yang terdiri dari kata-kata. Kemudian *array* ini akan dimasukkan ke dalam fungsi filtering() untuk menghilangkan kata-kata tidak penting kemudian dihasilkan suatu teks unik.

```
function bacaFile() {
   $handle = fopen ($this->getFullpath(), "r" )
   or exit("file gagal dibuka") ;

   while (!feof($handle)){
        $baca = fread ($handle , 1024000);
   }
   fclose($handle);
   return $baca;
}
```

Gambar 4.6 *Source code* Fungsi bacaFile ()

Pada fungsi bacaFile() pertama-tama dilakukanpembacaan terhadap path yang terdapat dokumen uji dan dokumen latihnya. Dimana path telah dideklarasikan sebelumnya. Kemudian membaca file dari awal sampai akhir.

```
function jumlahKata(){
  $data=explode(" ",$this->bacaFile());
  $jumArray=count($data);

  return $data;
}


function jumlahKalimat(){
   $data=explode(".",$this->bacaFile());
   $jumArray=count($data);
      for ($i=0;$i<$jumArray;$i++){
      $temp=$data[$i];
        if ($temp==''){
           $kalimat=array_shift($data);
        }
      }

   return count($data);
}
```

```
function jumlahParagraf(){
   $cleanFile=trim($this->bacaFile());
   $data=explode("\r\n",$cleanFile);
   $jumArray=count($data);

   for ($i=0;$i<$jumArray;$i++){
   $temp=$data[$i];
        if(empty($temp)|| isset($temp)){
           $kalimat=array_shift($data);
        }
   }
   return $jumArray;
}
```

Gambar 4.7 *Source code* Fungsi jumlahKata(), jumlahKalimat(), jumlahParagraf()

Untuk menghitung jumlah kata, hasil dari bacaFile tadi dipecah berdasarkan tanda spasi sehingga didapatkan jumlah kata pada teks tersebut.

```
function fileInfo() {
  $info=filesize($this->getFullpath());
  return $info;
}
```

Gambar 4.8 Source code fungsi infoFile()

Setelah mendapatkan informasi, tahap berikutnya yaitu *tokenizing* dilakukan pada fungsi tokenizingSubstring(). Proses yang dilakukan dalam fungsi tokenizingSubstring() yaitu *casefolding*, mengubah menjadi huruf kecil semua, menghilangkan tanda baca/simbol dan memecah teks menjadi potongan kata. Fungsi tokenizing() dapat dilihat pada gambar 4.9.

Setelah melalui tahap *tokenizing* kemudian dilakukan *filtering* pada fungsi filtering() yang terdapat pada gambar 4.10. Didalam fungsi ini dilakukan penghilangan kata-kata yang tidak penting. Kata-kata yang tidak penting berdasarkan pada jurnal Tala.

```
public function tokenizingSubstring ($string)  {
$a = 0;
$string     = strtolower($string);
$delimiter  = array('.',',','"',"'",'-
    ','/','{','}','+','_','!','@','#','$','%','^'
    ,'&','*','(',')','?','>','<','[',']','|','`',
    '~',';',':','=','\\',"\n","\r");

$teksTemp  = str_replace($delimiter," ",$string);
$temp      = explode(" ",trim($teksTemp));
```

Gambar 4.9 *Source code* fungsi tokenizingSubstring()

fungsi tokenizingSubstring yaitu merubah menjadi huruf kecil semua dengan menggunakan `strtolower` kemudian menghilangkan tanda baca dan simbol yang telah ditentukan dalam variabel bertipe *array*. Jika ditemukan maka simbol atau tanda baca pada teks akan dihapus.

```
public function filtering($string){
$query = mysql_query("SELECT * FROM m_stopword");

while ($row = @mysql_fetch_array($query)){
   $stopword[] = trim($row['stopword']);
}
   for($i=0;$i<=count($string);$i++):
      if(in_array($string[$i],$stopword))
      unset($string[$i]);
   endfor;
$result['array_kata'] = $string;
for($i=0;$i<count($temp);$i++){
     if($temp[$i]=='')
     unset($temp[$i]);
 }
        // Pengisian nilai array
        foreach ($temp as $word):
           if($word!=''){
             $result[$a] = trim($word);
             $a++;
           }
        endforeach;
```

```
foreach($string as $kata):

   $result['kalimat_unik']=$result['kalimat_unik'].tri
   m($kata);

endforeach;

return $result; }
```

Gambar 4.10 *Source code* fungsi filtering()

Pada fungsi *filtering* diatas, pertama memanggil kata tidak penting yang terdapat pada *database* kemudian disimpan dalam array. Kemudian membaca seluruh teks, jika ditemukan kata tidak penting maka akan dihapus, sehingga nantinya akan menghasilkan kalimat unik

Setelah dilakukan proses *filtering* akan ditambahkan proses *stemming*, yaitu suatu proses untuk mengubah bentuk kata menjadi kata dasar. *Stemming* yang digunakan berdasarkan penelitian yang dilkuakan oleh Arifin-Setiono. Gambar 4.5 adalah *source code* dari fungsi *stemming*(). Fungsi *stemming*() selengkapnya dapat dilihat pada lampiran.

4.3.3. Tahap *Parsing Kgram* dan Pembentukan *Hash Value*

Setelah tahap preprocessing akan dilakukan *parsing* teks menjadi k bagian dengan menggunakan *kgram*. Tahap ini dilakukan pada fungsi *kgram*Parsing() yang terdapat pada gambar 4.11.

Setelah terbentuk kumpulan substring dari proses *kgram*Parsing() akan dilakukan pembentukan *hash value* dengan fungsi hashing2() seperti pada gambar 4.12 sedangkan untuk modifikasi fungsi *hash*-nya dapat dilihat pada gambar 4.13

```
public function kgramParsing($string, $kgram){
$kgramAwal  = $kgram;
$jum        = strlen($string);
$temp       = 0;
$kgram_temp = 0;
$batas      = $kgramAwal-1;
for($i=0;$i<$jum-$batas;$i++) {
   $res[] = substr($string,$i,$kgramAwal);
   $temp  = $kgram;
   $kgram = $kgram+1;
}
return $res;   }
```

Gambar 4.11 *Source code* fungsi *kgram*Parsing()

Pada *source code* diatas digunakan untuk melakukan *parsing* dengan menggunakan fungsi `substr` yang dimulai dari i dan dipotong sebesar inputan dari *kgram*, maka akan dihasilkan kumpulan *array substring* hasil dari *parsing kgram.*

```
public function hashing2 ($kata,$posisi){
$jum_huruf = strlen($kata);
$n = 0;
$result['string'] = $kata;
$result['posisi'] = $posisi;
    for($i=$jum_huruf-1;$i>=0;$i--){
      $temp  = pow($this->basis , $i )* ord($kata[$n]);
      $result['hash_n'][] = $temp;
      $result['hash']     = $result['hash']+$temp;
      $result['ketemu']   = false;
      $n++;
     }
  $result['hash_mod'] = $result['hash']%$this->modulo;
  return $result;
}
```

Gambar 4.12 S*ource code* fungsi hashing2()

```
public function hashing2 ($kata,$posisi){
$jum_huruf = strlen($kata);
$n = 0;
$result['string'] = $kata;
$result['posisi'] = $posisi;
     for($i=$jum_huruf-1;$i>=0;$i--){
        $temp  = pow($this->basis , $i )* ord($kata[$n]);
        $result['hash_n'][] = $temp;
        $result['hash']     = $result['hash']+$temp;
        $result['ketemu']   = false;
        $n++;
     }

$result['hash_mod']= $result['hash']%$this->modulo;
$result['hash_div']= $result['hash']/$this->modulo;
$result['hash_div']=number_format($result['hash_div'],2);

return $result; }
```

Gambar 4.13 Sorce code fungsi hashing()

Fungsi diatas merupakan implementasi dari fungsi *hashing* yang telah dijelaskan sebelumnya pada bab 3.Perbedaan dari fungsi *hashing* diatas adalah dengan penambahan hasil bagi dari *modulo* yang akan digunakan pada pencocokan *string* nantinya.

4.3.4. Tahap *String Matching*

Setelah pembentukan nilai *hash* maka akan dilakukan pencocokan string. Fungsi yang digunakan untuk pencocokan *string* adalah stringMatching() dapat dilihat pada gambar 4.14. Sedangkan untuk modifikasi *string matching* dapat dilihat pada gambar 4.15

Fungsi stringMatching() pada gambar 4.15 digunakan untuk menemukan *string* yang sama. Apabila nilai *hash* dari kedua dokumen yang diuji sama maka, akan dilakukan pengecekan per karakter. Jika semua karakter sama, berarti string tersebut ditemukan. Sedangkan pada *string matching* yang dimodifikasi, melakukan pengecekan terhadap hasil bagi dan hasil modulonya.

```
function stringMatching($patternHash, $teksHash){
 $tempSama = 0;
for($a=0;$a<count($patternHash);$a++){
   for($b=0;$b<count($teksHash);$b++){

if($patternHash[$a]['hash_mod']==$teksHash[$b]['h
ash_mod']){
      for($k=0;$k<$this->kgram;$k++){
        if($patternHash[$a]['string'][$k]==$teksH
        ash[$b]['string'][$k]){
            $tempSama=$tempSama+1;
        }
          if ($k==$this->kgram-1){
             if ($tempSama==$this->kgram){
                 $patternHash[$a]['ketemu']='ya';
                     break 2;}
            else

            $patternHash[$a]['ketemu']='tidak';

          $tempSama=0;
                }
             }
           }
      else
       $patternHash[$a]['ketemu']='tidak';
     }
       $tempSama=0;
}
  return $patternHash;
}
```

Gambar 4.14 *Source code* fungsi stringMatching()

```
public function stringMatchImproved($pattern,$string){
for($a=0;$a<count($pattern);$a++){

for($b=0;$b<count($string);$b++){
     if(($pattern[$a]['hash_mod']==$string[$b]['hash_m
        od'])&&($pattern[$a]['hash_div']==$string[$b][
        'hash_div'])){

           $pattern[$a]['ketemu']='ya';
           break 1;
       }
     else
           $pattern[$a]['ketemu']='tidak';
     }
   }
return $pattern;
 }
```

Gambar 4.15. *Source code* fungsi stringMatchImproved()

4.3.5. Tahap Hitung *Similarity*

Setelah melakukan proses pencocokan ***string***, maka dilakukan tahap penghitungan nilai ***similarity***. Penghitungan persentase nilai ***similarity*** tersebut terdapat pada fungsi similarityNgram(), seperti pada gambar 4.16 berikut:

```
public function similarityNgram
($ketemu,$substringAsli,$substringUji){
     $result=
    (2*$ketemu)/($substringAsli+$substringUji))*100;
     return $result;
 }
```

Gambar 4.16 *Source code* fungsi similarityNgram()

Fungsi similarityNgram diatas adalah implementasi dari *Dice's Similarity Coefficient* yang telah dijelaskan pada bab 2. Fungsi ini digunakan untuk menetukan nilai ***similarity*** antara dua dokumen yang diuji.

4.4. Hasil Percobaan Sistem

4.4.1 Hasil Uji Coba Modulo

Tabel 4.1 dan tabel 4.2 adalah contoh hasil uji coba *modulo* terhadap masing-masing algoritma, dokumen uji yang digunakan adalah kode dokumen A dan dokumen latihnya dilakukan penghapusan sebanyak 20% kata yang berarti 80 % kata didalam teks tersebut adalah sama. Untuk tabel selengkapnya terdapat pada lampiran.

4.4.1.1. Hasil Uji *Modulo*: Algoritma Rabin-Karp

Tabel 4.1 Tabel uji *modulo* dengan *kgram*=1

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	1	13	88.0527540729	0.105367898941
2	1	23	88.0527540729	0.120038032532
3	1	41	88.0527540729	0.11904501915
4	1	53	88.0527540729	0.113399028778
5	1	71	88.0527540729	0.103160142899
6	1	97	88.0527540729	0.117776870728
7	1	101	88.0527540729	0.096480846405
8	1	151	88.0527540729	0.115622997284
9	1	173	88.0527540729	0.130659103394
10	1	257	88.0527540729	0.111675024033

4.4.1.2. Hasil Uji *Modulo*: Algoritma Rabin-Karp Modifikasi

Tabel 4.2 Tabel uji *modulo* dengan *kgram*=1

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	1	13	88.0527540729	0.125046014786
2	1	23	88.0527540729	0.123197793961
3	1	41	88.0527540729	0.113349199295
4	1	53	88.0527540729	0.131970882416
5	1	71	88.0527540729	0.148539066315
6	1	97	88.0527540729	0.140691995621
7	1	101	88.0527540729	0.104959011078
8	1	151	88.0527540729	0.136391162872
9	1	173	88.0527540729	0.122011899948
10	1	257	88.0527540729	0.125108957291

4.4.2 Hasil Uji Coba Nilai *Similarity* dan Waktu Proses

Pada uji coba ini terdapat 4 dokumen uji yang masing-masing mempunyai 8 dokumen latih menggunakan *kgram* 1 sampai 5 dengan algortima Rabin-Karp dan Rabin-Karp yang telah dimodifikasi serta menggunakan *stemming* dan *non-stemming.* Tabel 4.3. dan tabel 4.4. adalah contoh hasil uji coba terhadap kgram=1. Untuk data hasil percobaan selengkapnya terdapat pada lampiran.

Table 4.3. Hasil Pengujian dengan *kgram*=1 dan tanpa menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin – Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	0.53	100.00	0.33	100.00
2		80% sama	1686	0.37	78.66	0.52	78.66
3		60% sama	1196	0.26	54.54	0.33	54.54
4		40% sama	736	0.33	32.92	0.27	32.92
5		20% sama	264	0.23	9.36	0.41	9.36
6		Tukar 20%	2159	0.29	100.00	0.33	100.00
7		Tukar 40%	2159	0.34	100.00	0.31	100.00
8		Ganti 10%	2081	0.35	95.08	0.39	95.08
9	B	100% sama	1737	0.43	100.00	0.28	100.00
10		80% sama	1391	0.48	81.37	0.44	81.37
11		60% sama	994	0.44	55.73	0.24	55.73
12		40% sama	661	0.32	37.10	0.27	37.10
13		20% sama	297	0.28	15.85	0.23	15.85
14		Tukar 20%	1750	0.47	100.00	0.28	100.00
15		Tukar 40%	1750	0.32	100.00	0.38	100.00
16		Ganti 10%	1735	0.54	98.99	0.31	98.99

17	C	100% sama	4705	0.56	100.00	0.91	100.00
18		80% sama	3670	0.42	78.71	0.47	78.71
19		60% sama	2676	0.48	56.72	0.42	56.72
20		40% sama	1643	0.39	33.00	0.39	33.00
21		20% sama	689	0.41	12.89	0.30	12.89
22		Tukar 20%	4753	0.66	99.64	0.55	99.64
23		Tukar 40%	4753	0.54	99.12	0.56	99.12
24		Ganti 10%	4681	0.64	99.42	0.58	99.42
25	D	100% sama	13192	1.15	100.00	1.41	100.00
26		80% sama	10385	1.02	79.06	1.13	79.06
27		60% sama	7725	0.92	58.46	0.80	58.46
28		40% sama	4954	0.70	37.39	0.65	37.39
29		20% sama	2299	0.50	16.40	0.86	16.40
30		Tukar 20%	13311	1.26	99.48	1.31	99.48
31		Tukar 40%	13311	1.05	99.81	1.24	99.81
32		Ganti 10%	13193	1.43	100.00	1.48	100.00

Table 4.4. Hasil Pengujian dengan *kgram*=1 dan menggunakan ***stemming***

No	Kode Dok	Dok Latih	Size (bytes)	Rabin – Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	12.32	100.00	12.20	100.00
2		80% sama	1686	10.52	78.08	11.23	78.08
3		60% sama	1196	10.20	53.75	9.84	53.75
4		40% sama	736	8.36	33.17	9.22	33.17
5		20% sama	264	7.41	9.50	8.06	9.50
6		Tukar 20%	2159	11.31	100.00	11.88	100.00
7		Tukar 40%	2159	11.54	100.00	10.95	100.00
8		Ganti 10%	2081	10.53	99.58	10.25	99.58
9	B	100% sama	1737	6.79	100.00	6.27	100.00
10		80% sama	1391	5.95	80.63	6.50	80.63
11		60% sama	994	5.25	56.81	5.50	56.81
12		40% sama	661	5.15	36.63	4.78	36.63
13		20% sama	297	4.29	16.35	4.11	16.35
14		Tukar 20%	1750	6.53	100.00	6.45	100.00
15		Tukar 40%	1750	7.19	100.00	6.78	100.00
16		Ganti 10%	1735	6.14	97.58	8.23	97.58

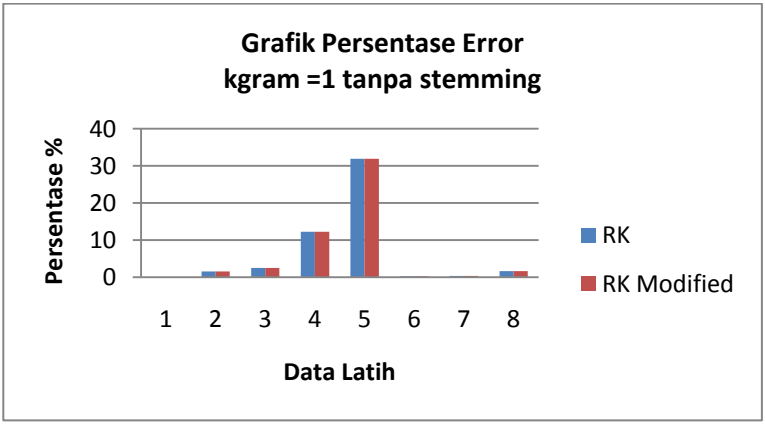
17	C	100% sama	4705	31.72	100.00	28.70	100.00
18		80% sama	3670	25.34	78.18	28.14	78.18
19		60% sama	2676	23.36	57.13	24.78	57.13
20		40% sama	1643	20.27	33.40	20.16	33.40
21		20% sama	689	17.12	13.33	17.87	13.33
22		Tukar 20%	4753	29.36	99.58	30.38	99.58
23		Tukar 40%	4753	28.55	99.20	30.19	99.20
24		Ganti 10%	4681	31.22	99.83	27.44	99.83
25	D	100% sama	13192	56.39	100.00	57.09	100.00
26		80% sama	10385	46.21	79.11	48.15	79.11
27		60% sama	7725	41.49	59.05	42.47	59.05
28		40% sama	4954	38.06	38.10	36.74	38.10
29		20% sama	2299	30.57	16.28	30.95	16.28
30		Tukar 20%	13311	55.87	99.48	52.61	99.48
31		Tukar 40%	13311	57.31	99.63	58.36	99.63
32		Ganti 10%	13193	57.77	100.00	55.53	100.00

Tabel diatas merupakan hasil perhitungan sistem terhadap algoritma Rabin-Karp dan algoritma Rabin-Karp yang telah dimodifikasi. Dapat dilihat bahwa nilai *similarity* yang dihasilkan kedua algoritma tersebut relatif sama, perbedaan yang cukup terlihat adalah pada waktu prosesnya (*running time*). Untuk data hasil percobaan selengkapnya dapat dilihat pada lampiran.

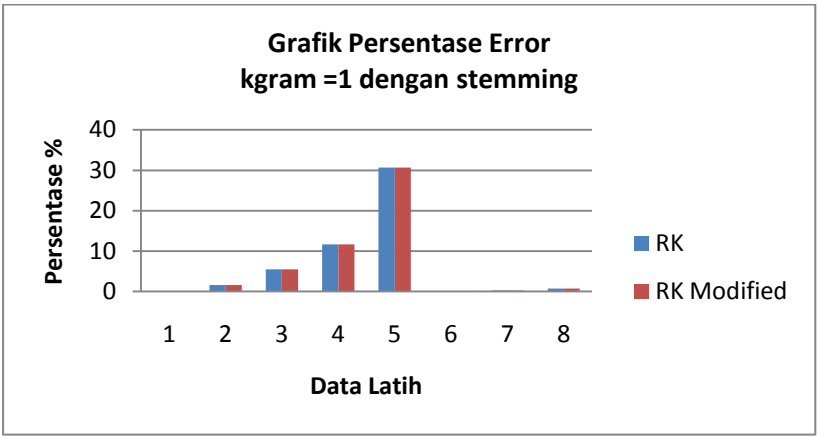
4.4.3 Hasil Uji Coba Persentase *Error*

Gambar 4.13 sampai 4.22 adalah grafik persentase *error* yang dihasilkan masing-masing algoritma, grafik dimulai dari *kgram*=1 hingga *kgram*=5

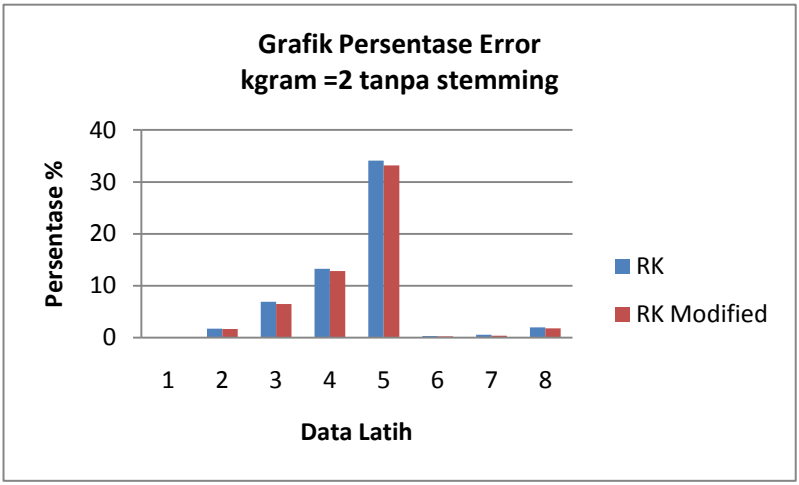
Keterangan data latih:
Data latih 1: 100% kata sama
Data latih 2: 80% kata sama
Data latih 3: 60% kata sama
Data latih 4: 40% kata sama
Data latih 5: 20% kata sama
Data latih 6: Tukar 20% kalimat
Data latih 7: Tukar 40% kalimat
Data latih 7: Ganti partikel kata 10%



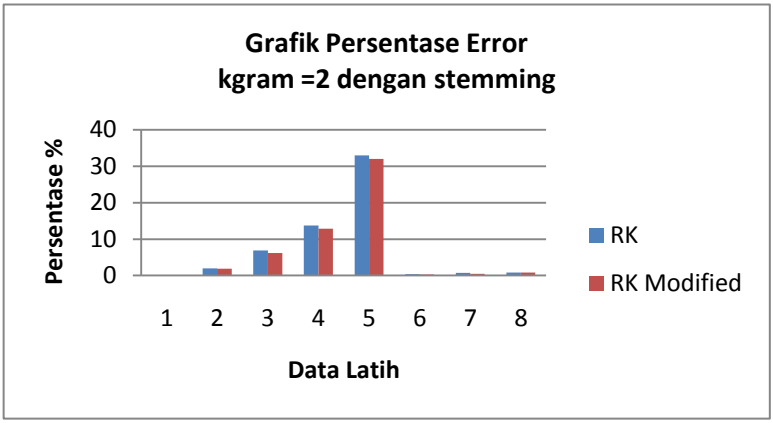
Gambar 4.13 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=1 tanpa *stemming*



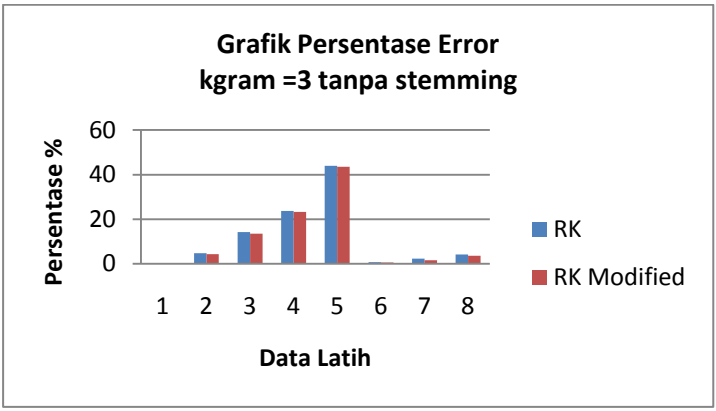
Gambar 4.14 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=1 menggunakan *stemming*



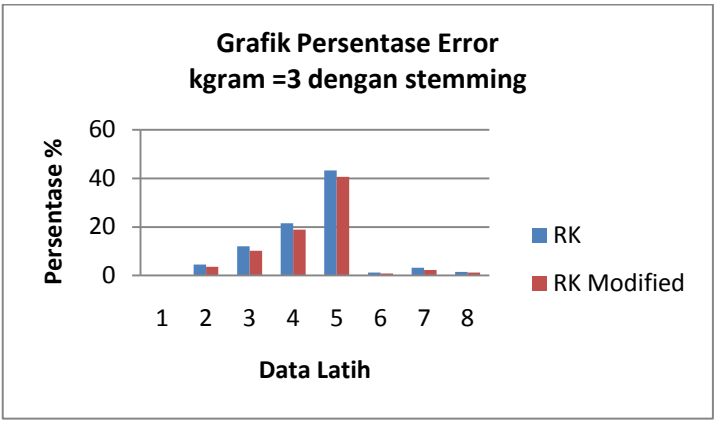
Gambar 4.15 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=1 tanpa *stemming*



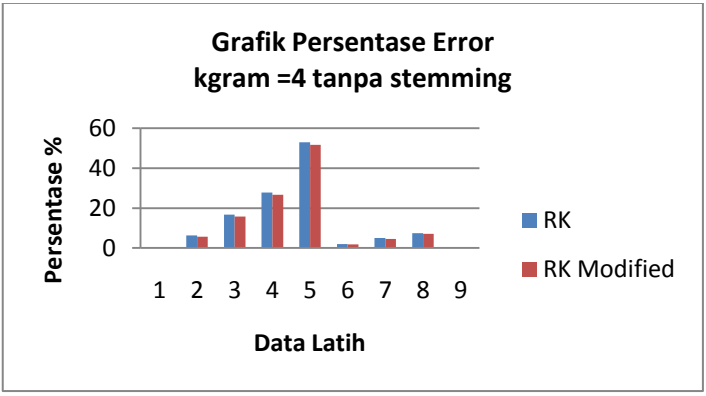
Gambar 4.16 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=2 menggunakan *stemming*



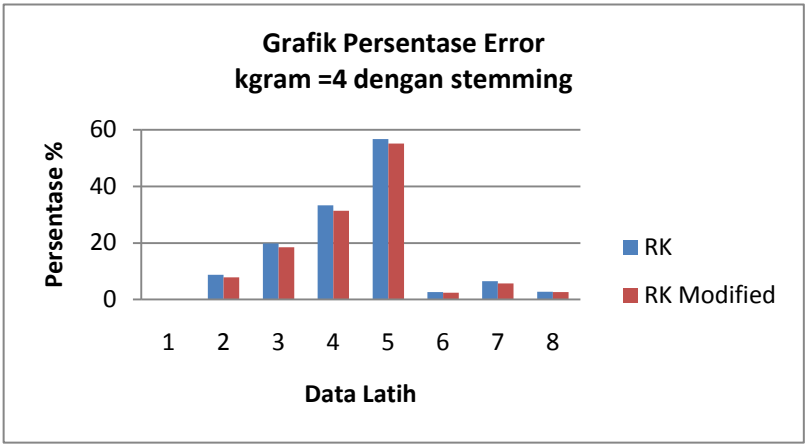
Gambar 4.18 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=3 tanpa *stemming*



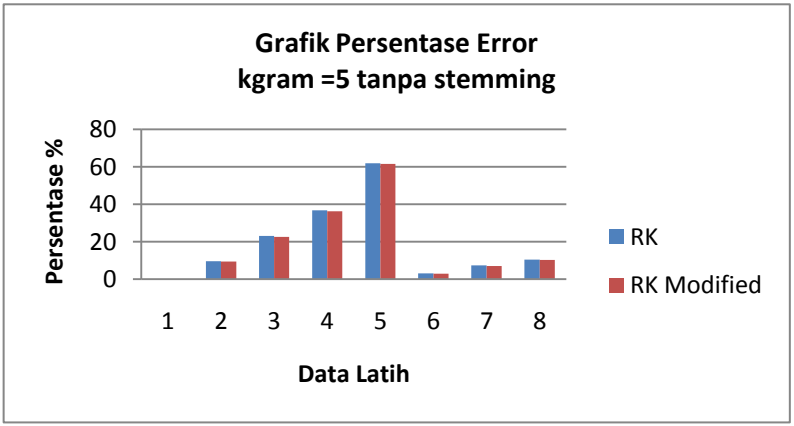
Gambar 4.17 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=3 menggunakan *stemming*



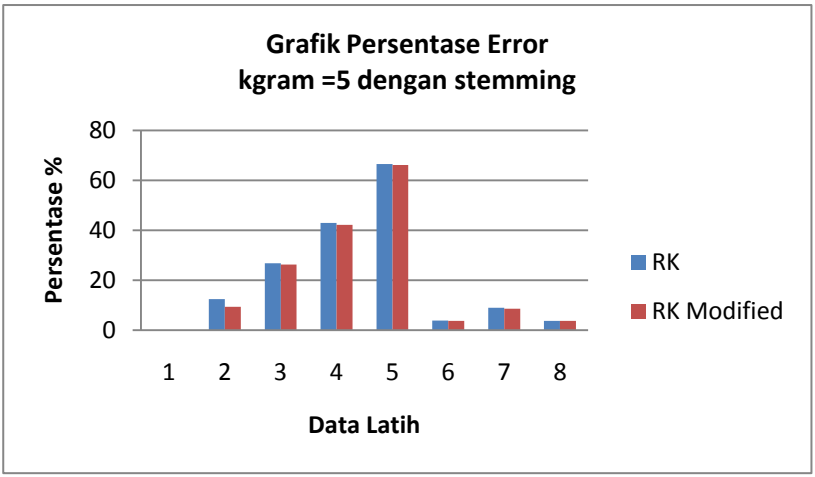
Gambar 4.19 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=4 tanpa *stemming*



Gambar 4.20 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=4 menggunakan *stemming*



Gambar 4.21 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=5 tanpa *stemming*

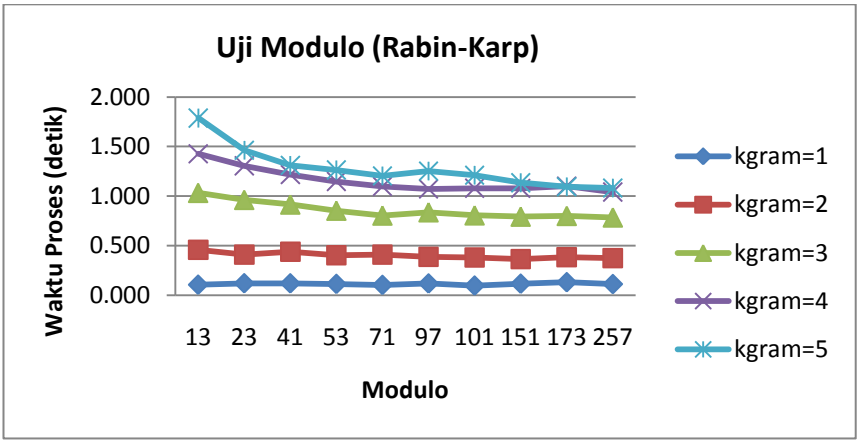


Gambar 4.22 Grafik perbandingan persentase *error* antar Rabin-Karp dan Rabin-Karp modifikasi, dengan *kgram*=5 menggunakan *stemming*

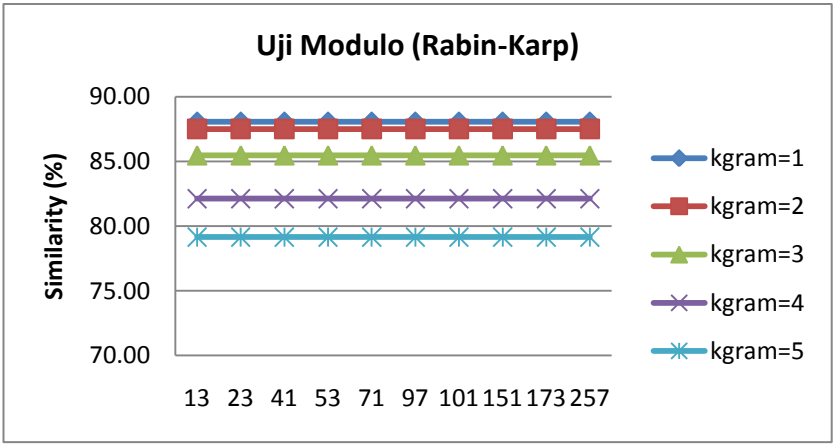
Grafik diatas merupakan perbandingan persentase *error* antara algoritma Rabin-Karp dengan Rabin-Karp yang telah dimodifikasi dapat dilihat bahwa persentase error paling besar terjadi pada data latih , sedangkan untuk data latih 1 mempunyai persentase error yang paling kecil yaitu 0%.

Dari grafik diatas terlihat bahwa persentase *error* berbanding lurus dengan perubahan / penghapusan kata. Semakin banyak kata yang dihapus semakin besar persentase *error*nya. Algoritma Rabin-Karp dan Rabin-Karp yang dimodifikasi mempunyai persentase *error* relatif sama. Tetapi algoritma Rabin-Karp yang dimodifikasi mempuyai persentase *error* yang lebih baik pada kasus tertentu

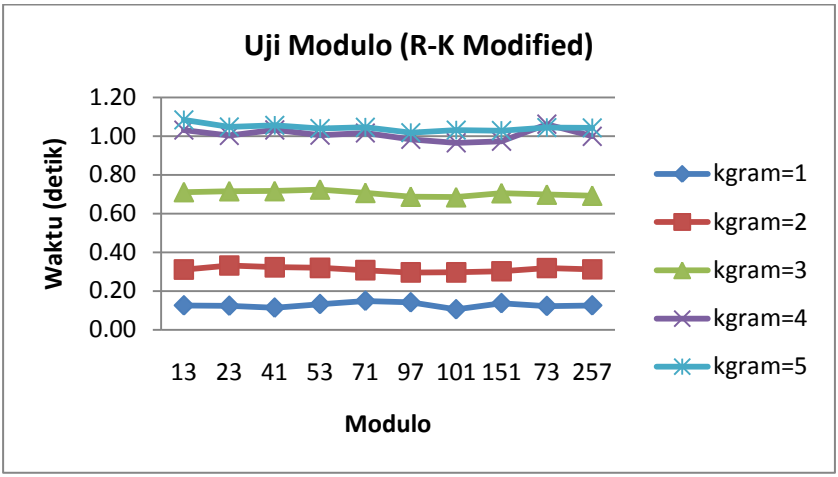
4.5. Analisa Hasil



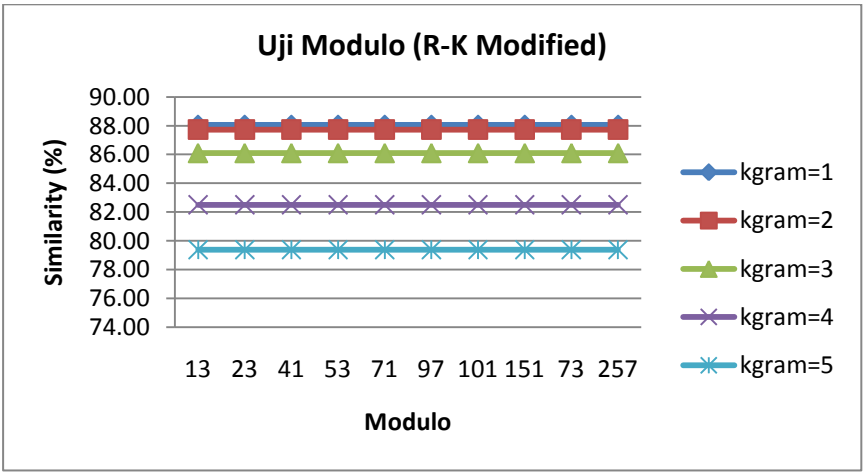
Gambar 4.22 Grafik perbandingan waktu dengan *kgram* 1-5 menggunakan algoritma Rabin-Karp



Gambar 4.23 Grafik perbandingan *similarity* dengan *kgram* 1-5 menggunakan algoritma Rabin-Karp



Gambar 4.24 Grafik perbandingan waktu dengan *kgram* 1-5 menggunakan algoritma Rabin-Karp dimodifikasi

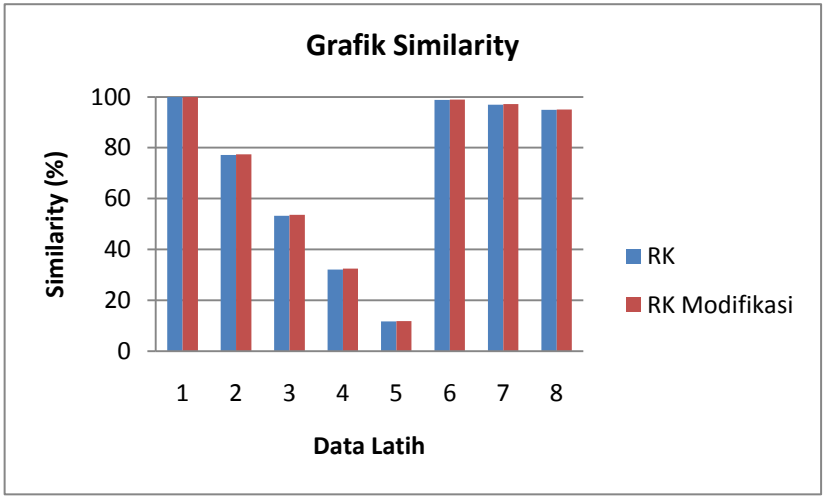


Gambar 4.25 Grafik perbandingan waktu dengan *kgram* 1-5 menggunakan algoritma Rabin-Karp dimodifikasi

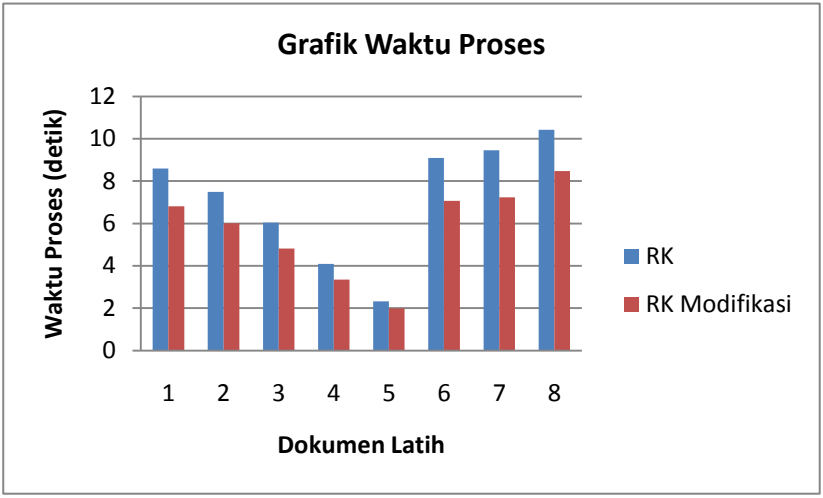
Dari grafik diatas dapat dilihat bahwa untuk *modulo* dengan menggunakan algoritma Rabin-Karp ada kecenderungan terjadi penurunan waktu proses, terutama dengan *kgram* lebih dari 2. Ketika

modulo=13 waktu yang dibutuhkan sistem lebih tinggi dibandingkan dengan menggunakan *modulo* yang lebih besar. Ketika *modulo*=71 dan seterusnya waktu proses sistem cenderung stabil tidak terjadi perubahan secara signifikan. Sedangkan perbandingan waktu proses terhadap modulo dengan menggunakan algoritma Rabin-Karp yang dimodifikasi waktu prosesnya stabil. Hal ini dikarenakan semakin tinggi modulonya akan menghasilkan *hash value* yang semakin bervariasi pada saat proses *hashing.* Sehingga pada algoritma Rabin-Karp akan mengurangi waktu untuk pengecekan karakter terhadap *hash value* yang sama. Sedangkan pada Rabin-Karp yang dimodifikasi, waktu yang digunakan cenderung stabil dikarenakan tidak terjadi pengecekan terhadap karakter.

Sedangkan untuk *similarity* baik menggunakan algoritma Rabin-Karp maupun Rabin-Karp yang telah dimodifikasi tidak terjadi perubahan nilai *similarity*, berarti berapapun *modulo* yang digunakan tidak mempengaruhi nilai *similarity*-nya.



Gambar 4.26 Grafik rata-rata nilai *similarity* algoritma Rabin-Karp dan Rabin-Karp Modifikasi



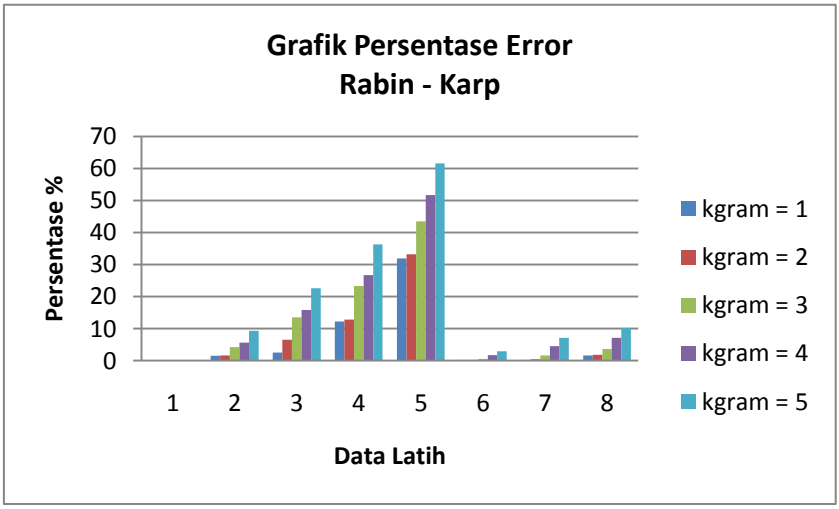
Gambar 4.27 Grafik rata-rata waktu proses algoritma Rabin-Karp dan Rabin-karp Modifikasi

Dari hasil percobaan yang dilakukan, pada gambar 4.26 algoritma Rabin-Karp yang dimodifikasi menghasilkan nilai *simalirity* yang hampir sama dibandingkan dengan dengan algoritma Rabin-Karp biasa. Sedangkan dari segi waktu proses algoritma Rabin-Karp modifikasi dilihat dari gambar 4. 27 mempunyai rata-rata waktu yang lebih baik dibandingkan dengan algoritma Rabin-Karp biasa, yang terlihat cukup signifikan adalah ketika data yang diuji cukup besar seperti pada percobaan dokumen C dan D yang mempunyai ukuran cukup besar. Hal ini dikarenakan algoritma Rabin-Karp modifikasi tidak memerlukan waktu tambahan untuk pengecekan karakter yang memiliki nilai hash yang sama, sedangkan pada algoritma Rabin-Karp biasa melakukan pengecekan karakter sehingga waktu proses menjadi lebih lama dibandingkan dengan algoritma Rabin-Karp modifikasi.

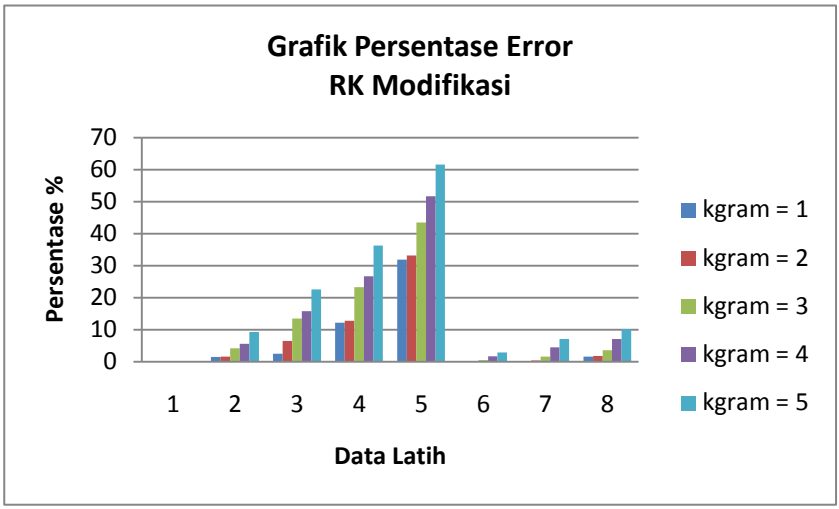
Setelah dilakukan percobaan dapat diketahui bahwa pemilihan *kgram* yang semakin kecil cenderung mempunyai tingkat akurasi nilai *similarity* yang lebih baik dibandingkan dengan menggunakan *kgram* yang lebih besar. Hal ini karena pada *kgram* yang lebih sedikit, *string* yang dipotong lebih kecil sehingga

kemungkinan untuk ditemukannya *string* yang sama semakin besar. Dengan semakin besarnya *kgram,* maka potongan *string* mengandung huruf yang lebih banyak dibandingkan dengan *kgram* yang lebih sedikit sehingga menyebabkan *string* yang ditemukan pun semakin berkurang.

Untuk penggunaan *stemming* membutuhkan waktu proses yang cukup tinggi dibandingkan tanpa menggunakan *stemming*, hal ini dikarenakan *stemming* yang digunakan melakukan pengecekan terhadap kata dasar yang terdapat pada database. Sedangkan untuk nilai *similarity* ternyata menghasilkan akurasi nilai *similarity* sedikit kurang baik dibandingkan dengan pengujian tanpa menggunakan *stemming*, tetapi penggunaan *stemming* menghasilkan akurasi nilai *similarity* yang lebih baik terhadap dokumen yang dimanipulasi dengan mengganti kata pasif menjadi aktif atau sebaliknya dan penambahan partikel pada kata, misalnya: "memberikan" diganti "diberkan" atau "mempunyai" diganti dengan "punya". Seperti pada dokumen latih yang diganti katanya sebanyak 10%.



Gambar 4.28 Grafik rata-rata persentase *error* menggunakan algoritma Rabin-Karp



Gambar 4.29 Grafik rata-rata persentase *error* menggunakan algoritma Rabin-Karp Modifikasi

Keterangan data latih:
Data latih 1: 100% kata sama
Data latih 2: 80% kata sama
Data latih 3: 60% kata sama
Data latih 4: 40% kata sama
Data latih 5: 20% kata sama
Data latih 6: Tukar 20% kalimat
Data latih 7: Tukar 40% kalimat
Data latih 8: Ganti partikel kata 10%

Dari grafik persentase *error* diatas dapat dilihat dokumen yang sama mempunyai persentase *error* 0% sedangkan persentase *error* tertinggi terjadi ketika perubahan kata sebanyak 80%. Dapat disimpulkan bahwa semakin banyak perubahan kata maka tingkat persentase *error*nya semakin besar, hal ini dikarenakan algoritma Rabin-Karp adalah algoritma pencocokan string dimulai dari kiri-kanan jika terjadi banyak perubahan kata seperti penambahan dan penghapusan kata maka substring *kgram* yang ditemukan pun semakin sedikit sehingga akurasinya pun menurun. Lain halnya jika

perubahan tersebut dalam level kalimat, persentase *error*nya tidak terlalu tinggi karena pada level kalimat terdiri dari banyak kata, sehingga jika ada perubahan sampai 40% pun tingkat persentase *error*-nya pun antara 0-7% saja.

Jika dibandingkan antara Rabin-Karp dengan Rabin-Karp yang telah dimodifikasi ternyata rata-rata persentase *error* yang dihasilkan hampir sama karena nilai *similarity* yang dihasilkan kedua algoritma diatas tidak jauh berbeda, perbedaan yang cukup terlihat adalah pada waktu prosesnya (*running time*). Hal ini dikarenakan algoritma Rabin-Karp yang dimodifikasi tidak memerlukan pengecekan tambahan ketika nilai *hash* yang dihasilkan sama

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Dari percobaan-percobaan yang telah dilakukan dapat disimpulkan bahwa:

1. Telah dibuat suatu sistem yang dapat digunakan untuk mendeteksi plagiarisme terhadap dokumen teks dengan menggunakan algoritma Rabin-Karp.
2. Algoritma Rabin-Karp biasa dan algoritma Rabin-Karp yang telah dimodifikasi mempunyai akurasi nilai *similarity* yang relatif sama. Tetapi algoritma Rabin-Karp yang dimodifikasi mempunyai rata-rata waktu proses yang lebih baik, terutama dokumen teks yang mempunyai *size*/ukuran file yang besar
3. *Kgram* yang semakin kecil menghasilkan akurasi nilai *similarity* yang lebih baik dibandingkan *kgram* yang lebih besar.
4. Persentase *error* yang dihasilkan kedua algoritma diatas relatif sama. Persentase *error* terkecil pada *kgram*=1. Persentase *error* berbanding lurus dengan perubahan jumlah kata. Semakin banyak perubahan pada teks tersebut maka persentase *error* yang dihasilkan semakin besar.
5. Penggunaan *stemming* berpengaruh pada akurasi nilai *similarity* yang dihasilkan. Dengan menggunakan *stemming* menghasilkan nilai yang cenderung kurang baik dibandingkan tanpa menggunakan *stemming*. Tetapi pada kasus tertentu seperti pengubahan bentuk kalimat algoritma Rabin-Karp yang disisipi *stemming* menghasilkan akurasi nilai *similarity* yang lebih baik.

5.2. Saran

Untuk penelitian lebih lanjut, disarankan penggunaan data uji dan data latih yang lebih bervariasi seperti pengubahan bentuk kalimat yang lebih banyak sehingga pengaruh penggunaan *stemming* dapat lebih akurat. Dan juga penggunaan rumus *hashing* yang lebih baik sehingga menghasilkan akurasi yang mungkin lebih baik.

DAFTAR PUSTAKA

Arifin, Agus Zainal dan Ari Setiono, Novan. *Klasifikasi Dokumen Berita Kejadian Berbahasa Indonesia dengan Algoritma Single Pass Clustering*. Institut Teknologi Sepuluh November (ITS). Surabaya

Andres, Nicolas. Christopher. Saloko, Hadi. 2008. *Penelaahan Algoritma Rabin-Karp dan Perbandingan Prosesnya dengan Algoritma Knut-Morris-Path*. Institut Teknologi Bandung (ITB). Bandung

Atmopawiro, Alsasian. 2006. *Pengkajian Dan Analisis Tiga Algoritma Efisien Rabin-Karp, Knuth-Morris-Pratt, Dan Boyer-Moore Dalam Pencarian Pola Dalam Suatu Teks*. Program Studi Teknik Informatika, Institut Teknologi Bandung.

Even-Zohar, Y. 2002. *Introduction to Text Mining*, Supercomputing.

Fernando, Hary. 2009. *Perbandingan dan Pengujian Beberapa Algoritma Pencocokan String*. Program Studi Teknik Informatika, Institut Teknologi Bandung (ITB). Bandung.

Firdaus, Hari Bagus. 2008. *Deteksi Plagiat Dokumen Menggunakan Algoritma Rabin-Karp*. Program Studi Teknik Informatika Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung (ITB). Bandung.

Hearst, M. 2003. *What is Text Mining?*. http : // www . sims . berkeley. edu / ~hears t/ text-mining.html.

Hultberg, Jens dan Helger, Joakim Poromaa. 2007. *Seminar course in Algorithms - Project report* .

Iyer, Parvati., Singh, Abhipsita. 2005. *Document Similarity Analysis for a Plagiarism Detection System*, 2nd Indian International Conference on Artificial Intelegence (IICAI-05), pp. 2534 – 2544.

KBBI, 1997: 775

Karp, Richard M.; Rabin, Michael O. 1987. *Efficient randomized pattern-matching algorithms*

Kosinov, Serhiy. 2002. *Evaluation of N-Grams Conflation Approach in Text-Based Information Retrieval*. University of Alberta. Canada

Mutiara, Benny; Agustina, Sinta. 2008. *Anti Plagiarsm Application with Algorithm Karp-Rabin at Thesis in Gunadarma University*. Gunadarma University. Depok, Indonesia

Ridhatillah, Ardini . 2003. *Dealing with Plagiarism in the Information System Research Community: A Look at Factors that Drive Plagiarism and Ways to Address Them*, MIS Quarterly; Vol. 27, No. 4, p. 511-532/December 2003

Schleimer, Saul; Wilkerson, Daniel, Aiken Alex. 2003. *Winnowing: Local Algorithms for Document Fingerprinting*. SIGMOD. San Diego, CA. 2003, June 9-12, 2003.

Singh, Rajendar Chillar dan Kochar, Baresjh. 2008. *RB-Matcher: String Matching technique*. World of Academy of Science, Engineering and technology.

Stein, B., Meyer, S. zu Eissen. 2006. *Near Similarity Search and PlagiarismAnalysis*, 29th Annual Conference of the German Classification Society(GfKl), Magdeburg, ISDN 1431-8814,pp. 430 – 437.

Tan, A. 1999. *Text Mining: The state of the art and the challenges*, In Proc of the Pacific Asia Conf on Knowledge Discovery and Data Mining PAKDD'99 workshop on Knowledge Discovery from Advanced Databases.

Triawati, Chandra. 2009. *Metode Pembobotan Statistical Concept Based untuk Klastering dan Kategorisasi Dokumen Berbahasa Indonesia.* Institut Teknologi Telkom. Bandung.

LAMPIRAN

Lampiran I: Daftar Stop Word

ada	bagai	bersama-sama	di	ialah
adalah	bagaikan	betulkah	dia	ibarat
adanya	bagaimana	biasa	dialah	ingin
adapun	bagaimanakah	biasanya	diantara	inginkah
agak	bagaimanapun	bila	diantaranya	inginkan
agaknya	bagi	bilakah	dikarenakan	ini
agar	bahkan	bisa	dini	inikah
akan	bahwa	bisakah	diri	inilah
akankah	bahwasanya	boleh	dirinya	itu
akhirnya	banyak	bolehkah	disini	itukah
aku	beberapa	bolehlah	disinilah	itulah
akulah	begini	buat	dong	jangan
amat	beginian	bukan	dulu	jangankan
amatlah	beginikah	bukankah	enggak	janganlah
anda	beginilah	bukanlah	enggaknya	jika
andalah	begitu	bukannya	entah	jikalau
antar	begitukah	cuma	entahlah	juga
antara	begitulah	dahulu	hal	justru
antaranya	begitupun	dalam	hampir	kala
apa	belum	dan	hanya	kalau
apaan	belumlah	dapat	hanyalah	kalaulah
apabila	berapa	dari	harus	kalaupun
apakah	berapakah	daripada	haruslah	kalian
apalagi	berapalah	dekat	harusnya	ialah
apatah	berapapun	demi	hendak	ibarat
atau	berkali-kali	demikian	hendaklah	ingin
ataukah	bermacam	demikianlah	hendaknya	inginkah
ataupun	bermacam-macam	dengan	hingga	inginkan
	bersama	depan	ia	ini

inikah	kala	lagi	mengapa	saat
inilah	kalau	lagian	mereka	saatnya
itu	kalaulah	lah	merekalah	saja
itukah	kalaupun	lain	merupakan	sajalah
itulah	kalian	lainnya	meski	saling
jangan	kami	lalu	meskipun	sama
jangankan	kamilah	lama	mungkin	sama-sama
janganlah	kamu	lamanya	mungkinkah	sambil
jika	kamulah	lebih	nah	sampai
jikalau	kan	macam	namun	sana
juga	kapan	maka	nanti	sangat
justru	kapankah	makanya	nantinya	sangatlah
kala	kapanpun	makin	nyaris	saya
kalau	karena	malah	oleh	sayalah
kalaulah	karenanya	malahan	olehnya	se
kalaupun	ke	mampu	pada	sebab
kalian	kecil	mampukah	padahal	sebabnya
inikah	kemudian	mana	padanya	sebagai
inilah	kenapa	manakala	paling	sebagaimana
itu	kepada	manalagi	pantas	sebagainya
itukah	kepadanya	masih	para	sebaliknya
itulah	ketika	masihkah	pasti	sebanyak
jangan	khususnya	masing	pastilah	sebegini
jangankan	kini	masing-masing	per	sebegitu
janganlah	kinilah	mau	percuma	sebelum
jika	kiranya	maupun	pernah	sebelumnya
jikalau	kita	melainkan	pula	sebenarnya
juga	kitalah	melalui	pun	seberapa
justru	kok	memang	rupanya	sebetulnya

sebisanya	selama	sesama	telah
sebuah	selama-lamanya	sesegera	tentang
sedang	selamanya	sesekali	tentu
sedangkan	seluruh	seseorang	tentulah
sedemikian	seluruhnya	sesuatu	tentunya
sedikit	semacam	sesuatunya	terdiri
sedikitnya	semakin	sesudah	terhadap
segala	semasih	sesudahnya	terhadapnya
segalanya	semaunya	setelah	terlalu
segera	sementara	seterusnya	terlebih
seharusnya	sempat	setiap	tersebut
sehingga	semua	setidaknya	tersebutlah
sejak	semuanya	setidak-tidaknya	tertentu
sejenak	semula	sewaktu	tetapi
sekali	sendiri	siapa	tiap
sekalian	sendirinya	siapakah	tidak
sekaligus	seolah	siapapun	tidakkah
sekali-kali	seolah-olah	sini	tidaklah
sekalipun	seorang	sinilah	toh
sekarang	sepanjang	suatu	waduh
sekarang	sepantasnya	sudah	wah
seketika	sepantasnyalah	sudahkah	wahai
sekiranya	seperti	sudahlah	walau
sekitar	sepertinya	supaya	walaupun
sekitarnya	sering	tadi	wong
sela	seringnya	tadinya	yaitu
selain	serta	tak	yakni
selaku	serupa	tanpa	yang
selalu	sesaat	tapi	

Lampiran II: Tabel Hasil Uji Modulo

Tabel 5.1 Tabel uji *modulo* dengan *kgram*=2

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	2	13	87.5	0.457693099976
2	2	23	87.5	0.410396099091
3	2	41	87.5	0.438357114792
4	2	53	87.5	0.402698993683
5	2	71	87.5	0.409835100174
6	2	97	87.5	0.387275934219
7	2	101	87.5	0.379792928696
8	2	151	87.5	0.364549160004
9	2	173	87.5	0.383480072021
10	2	257	87.5	0.374601125717

Tabel 5.2 Tabel uji *modulo* dengan *kgram*=3

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	3	13	85.4700854701	1.03745412827
2	3	23	85.4700854701	0.962348937988
3	3	41	85.4700854701	0.915410041809
4	3	53	85.4700854701	0.853345870972
5	3	71	85.4700854701	0.804033041000
6	3	97	85.4700854701	0.836989879608
7	3	101	85.4700854701	0.805838823318
8	3	151	85.4700854701	0.792943954468
9	3	173	85.4700854701	0.800297021866
10	3	257	85.4700854701	0.784543037415

Tabel 5.3 Tabel uji *modulo* dengan *kgram*=4

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	4	13	82.1150855365	1.42620515823
2	4	23	82.1150855365	1.30556511879
3	4	41	82.1150855365	1.21692109108
4	4	53	82.1150855365	1.1474840641
5	4	71	82.1150855365	1.09865498543
6	4	97	82.1150855365	1.07171607018
7	4	101	82.1150855365	1.07976913452
8	4	151	82.1150855365	1.07821202278
9	4	173	82.1150855365	1.10117793083
10	4	257	82.1150855365	1.04110503197

Tabel 5.4 Tabel uji *modulo* dengan *kgram*=5

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	5	13	79.1439688716	1.7872710228
2	5	23	79.1439688716	1.46354007721
3	5	41	79.1439688716	1.31051707268
4	5	53	79.1439688716	1.26388216019
5	5	71	79.1439688716	1.2033469677
6	5	97	79.1439688716	1.25306916237
7	5	101	79.1439688716	1.21013593674
8	5	151	79.1439688716	1.13524794579
9	5	173	79.1439688716	1.09438896179
10	5	257	79.1439688716	1.08194589615

Lampiran III: Hasil Uji *Modulo*: Algoritm Rabin-Karp dimodifikasi

Tabel 5.5 Tabel uji *modulo* dengan *kgram*=2

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	2	13	87.7329192547	0.310649871826
2	2	23	87.7329192547	0.331465005875
3	2	41	87.7329192547	0.322463989258
4	2	53	87.7329192547	0.319169998169
5	2	71	87.7329192547	0.307395935059
6	2	97	87.7329192547	0.295697927475
7	2	101	87.7329192547	0.296142101288
8	2	151	87.7329192547	0.301293849945
9	2	173	87.7329192547	0.318148851395
10	2	257	87.7329192547	0.312112092972

Tabel 5.6 Tabel uji *modulo* dengan *kgram*=3

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	3	13	86.0916860917	0.710844039917
2	3	23	86.0916860917	0.715488910675
3	3	41	86.0916860917	0.715856790543
4	3	53	86.0916860917	0.724056005478
5	3	71	86.0916860917	0.706813812256
6	3	97	86.0916860917	0.687486886978
7	3	101	86.0916860917	0.684508085251
8	3	151	86.0916860917	0.705301046371
9	3	173	86.0916860917	0.697958946228
10	3	257	86.0916860917	0.692495107651

Tabel 5.7 Tabel uji *modulo* dengan *kgram*=4

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	4	13	82.5038880249	1.03022003174
2	4	23	82.5038880249	1.00293302536
3	4	41	82.5038880249	1.03157496452
4	4	53	82.5038880249	1.00471496582
5	4	71	82.5038880249	1.01612114906
6	4	97	82.5038880249	0.983326196671
7	4	101	82.5038880249	0.965177059174
8	4	151	82.5038880249	0.973064899445
9	4	173	82.5038880249	1.06111192703
10	4	257	82.5038880249	0.998518943787

Tabel 5.8 Tabel uji *modulo* dengan *kgram*=5

No	*KGRAM*	MODULO	SIMILARITY (%)	WAKTU PROSES (detik)
1	5	13	79.3774319066	1.08374500275
2	5	23	79.3774319066	1.04713320732
3	5	41	79.3774319066	1.05570197105
4	5	53	79.3774319066	1.03943300247
5	5	71	79.3774319066	1.04500985146
6	5	97	79.3774319066	1.01805114746
7	5	101	79.3774319066	1.03089499474
8	5	151	79.3774319066	1.02822804451
9	5	173	79.3774319066	1.04373288155
10	5	257	79.3774319066	1.04189801216

Lampiran IV: Hasil Percobaan Pengecekan Plagiarisme Dokumen menggunakan Algoritma Rabin-Karp dan Rabin-Karp yang Dimodifikasi

Table 5.9 Hasil Pengujian dengan *kgram*=2 dan tanpa menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	0.66	100.00	0.97	100.00
2		80% sama	1686	0.68	78.16	0.50	78.36
3		60% sama	1196	0.86	53.81	0.43	54.09
4		40% sama	736	0.44	32.18	0.63	32.45
5		20% sama	264	0.33	8.74	0.33	8.95
6		Tukar 20%	2159	0.74	100.00	0.59	100.00
7		Tukar 40%	2159	0.88	99.65	0.57	100.00
8		Ganti 10%	2081	1.21	94.17	0.64	94.66
9	B	100% sama	1737	0.77	100.00	0.50	100.00
10		80% sama	1391	0.53	81.10	0.49	81.27
11		60% sama	994	0.42	55.02	0.44	55.44
12		40% sama	661	0.57	36.88	0.55	36.96
13		20% sama	297	0.46	15.44	0.43	15.61
14		Tukar 20%	1750	0.94	100.00	0.60	100.00
15		Tukar 40%	1750	0.60	99.58	0.54	99.75
16		Ganti 10%	1735	0.93	98.57	0.52	98.65

17	C	100% sama	4705	2.41	100.00	1.99	100.00
18		80% sama	3670	2.16	78.49	1.39	78.58
19		60% sama	2676	1.69	56.28	1.09	56.52
20		40% sama	1643	1.20	32.62	0.73	32.83
21		20% sama	689	0.63	12.32	0.79	12.56
22		Tukar 20%	4753	3.19	99.45	1.79	99.51
23		Tukar 40%	4753	2.50	98.82	1.90	98.94
24		Ganti 10%	4681	2.48	99.27	1.94	99.39
25	D	100% sama	13192	7.41	100.00	5.40	100.00
26		80% sama	10385	6.71	78.97	4.27	79.00
27		60% sama	7725	4.85	58.27	3.52	58.38
28		40% sama	4954	3.50	37.10	2.38	37.20
29		20% sama	2299	1.98	16.23	1.49	16.32
30		Tukar 20%	13311	7.65	99.40	5.15	99.45
31		Tukar 40%	13311	8.08	99.78	5.36	99.78
32		Ganti 10%	13193	7.43	100.00	5.56	100.00

Table 5.10 Hasil Pengujian dengan *kgram*=2 dan menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	12.11	100.00	10.65	100.00
2		80% sama	1686	12.14	77.23	11.15	77.48
3		60% sama	1196	11.51	52.63	9.58	53.13
4		40% sama	736	9.07	31.69	9.18	32.28
5		20% sama	264	7.98	8.84	7.02	9.09
6		Tukar 20%	2159	13.55	99.83	11.27	99.92
7		Tukar 40%	2159	11.74	99.17	11.99	99.75
8		Ganti 10%	2081	10.21	99.42	10.43	99.42
9	B	100% sama	1737	7.72	100.00	6.99	100.00
10		80% sama	1391	6.37	80.30	6.46	80.40
11		60% sama	994	6.04	55.66	5.80	56.36
12		40% sama	661	5.06	35.66	5.00	36.06
13		20% sama	297	3.86	15.96	4.71	16.16
14		Tukar 20%	1750	6.67	100.00	6.74	100.00
15		Tukar 40%	1750	6.84	99.49	6.30	99.60
16		Ganti 10%	1735	6.40	97.58	6.89	97.58

17	C	100% sama	4705	32.44	100.00	28.60	100.00
18		80% sama	3670	29.55	77.86	25.48	77.97
19		60% sama	2676	26.69	56.46	23.28	56.80
20		40% sama	1643	22.09	32.86	20.87	33.14
21		20% sama	689	18.88	12.74	17.90	12.95
22		Tukar 20%	4753	33.80	99.37	32.18	99.44
23		Tukar 40%	4753	30.29	98.71	31.60	98.99
24		Ganti 10%	4681	29.61	99.79	32.56	99.79
25	D	100% sama	13192	60.49	100.00	55.98	100.00
26		80% sama	10385	54.79	78.97	55.07	79.03
27		60% sama	7725	49.43	58.78	48.55	58.90
28		40% sama	4954	40.53	37.78	41.23	37.87
29		20% sama	2299	32.95	16.09	32.58	16.19
30		Tukar 20%	13311	59.09	99.42	56.47	99.45
31		Tukar 40%	13311	55.24	99.56	57.13	99.60
32		Ganti 10%	13193	59.45	100.00	56.97	100.00

Table 5.11 Hasil Pengujian dengan *kgram*=3 dan tanpa menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	1.00	100.00	0.87	100.00
2		80% sama	1686	0.88	76.34	0.77	76.89
3		60% sama	1196	0.71	51.35	0.62	52.19
4		40% sama	736	0.53	29.56	0.48	30.12
5		20% sama	264	0.33	6.73	0.31	7.08
6		Tukar 20%	2159	0.96	99.79	0.85	99.79
7		Tukar 40%	2159	1.02	97.43	0.87	98.13
8		Ganti 10%	2081	1.97	90.22	1.36	91.05
9	B	100% sama	1737	2.28	100.00	0.86	100.00
10		80% sama	1391	1.89	79.14	0.73	79.81
11		60% sama	994	1.52	52.53	0.65	53.38
12		40% sama	661	1.19	34.12	0.49	34.46
13		20% sama	297	0.70	13.68	0.36	13.85
14		Tukar 20%	1750	1.98	99.32	0.87	99.75
15		Tukar 40%	1750	1.93	97.30	0.97	98.23
16		Ganti 10%	1735	1.18	95.44	0.95	95.95

17	C	100% sama	4705	10.33	100.00	4.25	100.00
18		80% sama	3670	5.31	77.03	3.52	77.63
19		60% sama	2676	7.49	54.08	2.64	54.93
20		40% sama	1643	5.57	30.65	1.95	31.38
21		20% sama	689	2.90	10.80	1.07	11.32
22		Tukar 20%	4753	10.39	98.66	4.34	98.97
23		Tukar 40%	4753	10.32	97.06	4.31	97.81
24		Ganti 10%	4681	8.14	97.97	5.56	98.54
25	D	100% sama	13192	24.93	100	17.34	100
26		80% sama	10385	19.5	87.41	14.34	87.88
27		60% sama	7725	16.56	72.11	11.16	72.94
28		40% sama	4954	7.84	52.19	5.37	53.21
29		20% sama	2299	5.96	26.34	4.02	27.08
30		Tukar 20%	13311	24.45	99.27	16.98	99.55
31		Tukar 40%	13311	25.24	98.86	16.82	99.44
32		Ganti 10%	13193	32.9	99.76	22.4	99.89

Table 5.12 Hasil Pengujian dengan *kgram*=3 dan menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	6.39	100.00	6.31	100.00
2		80% sama	1686	5.82	74.54	5.74	75.38
3		60% sama	1196	5.36	48.66	5.24	49.92
4		40% sama	736	4.56	27.63	4.52	28.80
5		20% sama	264	3.90	5.93	3.86	6.34
6		Tukar 20%	2159	6.32	99.42	6.26	99.58
7		Tukar 40%	2159	6.45	96.58	6.35	97.33
8		Ganti 10%	2081	12.06	98.25	13.14	98.41
9	B	100% sama	1737	6.88	100.00	9.86	100.00
10		80% sama	1391	5.90	77.25	8.10	78.16
11		60% sama	994	5.79	51.87	6.99	52.98
12		40% sama	661	4.39	32.15	5.71	33.06
13		20% sama	297	3.58	13.85	5.12	14.26
14		Tukar 20%	1750	7.04	98.69	9.01	99.29
15		Tukar 40%	1750	6.88	96.56	7.59	97.27
16		Ganti 10%	1735	8.14	96.46	7.87	96.97

17	C	100% sama	4705	31.08	100.00	33.43	100.00
18		80% sama	3670	27.75	75.66	28.61	76.43
19		60% sama	2676	24.31	53.38	26.22	54.63
20		40% sama	1643	20.97	29.91	23.14	31.02
21		20% sama	689	15.58	10.69	18.25	11.53
22		Tukar 20%	4753	31.57	98.29	35.15	98.85
23		Tukar 40%	4753	30.88	96.24	36.60	97.49
24		Ganti 10%	4681	34.62	99.16	32.90	99.51
25	D	100% sama	13192	63.49	100.00	59.48	100.00
26		80% sama	10385	56.07	77.93	51.84	78.49
27		60% sama	7725	51.29	57.10	45.90	58.06
28		40% sama	4954	39.12	35.93	36.63	36.93
29		20% sama	2299	30.44	14.86	37.61	15.38
30		Tukar 20%	13311	65.03	98.74	73.35	99.12
31		Tukar 40%	13311	66.65	98.02	58.35	98.85
32		Ganti 10%	13193	80.75	100.00	70.30	100.00

Table 5.13 Hasil Pengujian dengan *kgram*=4 dan tanpa menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	1.23	100.00	1.17	100.00
2		80% sama	1686	1.11	73.33	1.03	73.68
3		60% sama	1196	0.89	47.22	0.83	47.50
4		40% sama	736	0.64	25.63	0.60	26.11
5		20% sama	264	0.38	4.86	0.37	4.93
6		Tukar 20%	2159	1.25	99.31	1.19	99.31
7		Tukar 40%	2159	1.27	95.07	1.19	95.35
8		Ganti 10%	2081	2.44	85.63	1.82	85.97
9	B	100% sama	1737	2.09	100.00	1.21	100.00
10		80% sama	1391	2.05	76.42	1.40	76.92
11		60% sama	994	1.64	48.44	1.31	49.03
12		40% sama	661	1.28	30.09	0.70	30.35
13		20% sama	297	0.77	10.65	0.48	10.82
14		Tukar 20%	1750	2.19	97.80	1.32	97.97
15		Tukar 40%	1750	2.32	94.34	1.32	94.93
16		Ganti 10%	1735	1.65	89.86	1.80	90.28

17	C	100% sama	4705	9.99	100.00	8.46	100.00
18		80% sama	3670	9.21	73.98	8.07	74.47
19		60% sama	2676	6.08	50.18	5.26	50.79
20		40% sama	1643	4.52	26.90	3.63	27.38
21		20% sama	689	2.32	8.56	1.67	8.96
22		Tukar 20%	4753	10.62	97.42	8.52	97.57
23		Tukar 40%	4753	10.71	94.41	7.80	94.84
24		Ganti 10%	4681	10.55	95.87	9.23	96.24
25	D	100% sama	13192	40.89	100.00	34.29	100.00
26		80% sama	10385	40.58	76.25	29.90	76.80
27		60% sama	7725	29.19	54.11	23.91	54.84
28		40% sama	4954	20.15	32.79	16.30	33.47
29		20% sama	2299	11.08	13.50	8.41	13.91
30		Tukar 20%	13311	43.13	97.87	33.93	98.29
31		Tukar 40%	13311	47.08	96.22	36.58	96.86
32		Ganti 10%	13193	51.37	98.72	40.35	98.89

Table 5.14 Hasil Pengujian dengan *kgram*=4 dan menggunakan ***stemming***

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	6.39	100.00	6.41	100.00
2		80% sama	1686	5.85	71.18	5.83	71.76
3		60% sama	1196	5.40	43.61	5.33	44.11
4		40% sama	736	4.57	22.56	4.56	23.48
5		20% sama	264	3.87	3.93	3.85	4.09
6		Tukar 20%	2159	6.44	98.58	6.39	98.58
7		Tukar 40%	2159	6.52	93.90	6.46	94.49
8		Ganti 10%	2081	12.30	97.24	12.41	97.33
9	B	100% sama	1737	6.66	100.00	7.98	100.00
10		80% sama	1391	6.15	73.68	6.36	74.39
11		60% sama	994	5.17	46.66	5.26	47.57
12		40% sama	661	4.57	26.92	5.02	27.53
13		20% sama	297	3.66	10.02	3.84	10.22
14		Tukar 20%	1750	6.79	97.17	7.15	97.47
15		Tukar 40%	1750	6.84	92.81	6.83	93.52
16		Ganti 10%	1735	7.62	93.93	7.37	94.03

17	C	100% sama	4705	33.31	100.00	33.15	100.00
18		80% sama	3670	31.00	71.82	26.69	72.52
19		60% sama	2676	23.33	48.87	24.54	49.63
20		40% sama	1643	20.46	25.15	20.75	25.77
21		20% sama	689	15.50	8.12	16.15	8.46
22		Tukar 20%	4753	30.51	96.59	31.19	96.83
23		Tukar 40%	4753	32.71	92.65	31.92	93.38
24		Ganti 10%	4681	39.39	98.22	37.22	98.57
25	D	100% sama	13192	78.90	100.00	68.33	100.00
26		80% sama	10385	70.62	75.37	64.27	76.25
27		60% sama	7725	61.57	53.22	58.12	54.38
28		40% sama	4954	47.53	32.04	48.50	32.99
29		20% sama	2299	34.16	12.49	38.95	13.09
30		Tukar 20%	13311	79.94	96.95	74.80	97.54
31		Tukar 40%	13311	80.09	94.72	75.58	95.75
32		Ganti 10%	13193	96.37	99.37	91.57	99.63

Table 5.15 Hasil Pengujian dengan *kgram*=5 dan tanpa menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	2.10	100.00	1.87	100.00
2		80% sama	1686	1.67	70.67	1.78	70.88
3		60% sama	1196	1.75	43.36	1.61	43.64
4		40% sama	736	1.11	21.75	0.98	21.89
5		20% sama	264	0.49	3.47	0.74	3.47
6		Tukar 20%	2159	2.05	98.75	1.94	98.75
7		Tukar 40%	2159	2.21	93.40	1.84	93.54
8		Ganti 10%	2081	2.39	81.72	2.54	81.93
9	B	100% sama	1737	2.75	100.00	1.63	100.00
10		80% sama	1391	2.22	73.77	1.15	73.86
11		60% sama	994	1.91	44.67	0.98	44.75
12		40% sama	661	1.43	26.23	1.29	26.31
13		20% sama	297	0.86	8.46	0.55	8.46
14		Tukar 20%	1750	2.39	96.70	1.33	96.79
15		Tukar 40%	1750	2.47	92.30	1.65	92.64
16		Ganti 10%	1735	1.72	85.45	2.47	85.70

17	C	100% sama	4705	10.05	100.00	7.98	100.00
18		80% sama	3670	8.39	70.85	8.76	71.09
19		60% sama	2676	7.03	46.10	6.28	46.25
20		40% sama	1643	4.48	23.63	4.32	23.87
21		20% sama	689	2.46	7.01	1.89	7.11
22		Tukar 20%	4753	10.35	96.11	8.74	96.17
23		Tukar 40%	4753	10.54	91.77	8.70	91.89
24		Ganti 10%	4681	12.45	93.68	12.06	93.87
25	D	100% sama	13192	50.38	100.00	44.39	100.00
26		80% sama	10385	44.75	73.90	39.64	74.21
27		60% sama	7725	36.25	50.67	33.89	51.09
28		40% sama	4954	25.56	29.52	25.20	29.83
29		20% sama	2299	13.33	11.46	15.17	11.71
30		Tukar 20%	13311	56.96	96.42	50.79	96.67
31		Tukar 40%	13311	59.59	93.36	51.64	93.67
32		Ganti 10%	13193	66.83	97.34	57.51	97.50

Table 5.16 Hasil Pengujian dengan *kgram*=5 dan menggunakan *stemming*

No	Kode Dok	Dok Latih	Size (bytes)	Rabin - Karp		R-K Modified	
				t(s)	s(%)	t(s)	s(%)
1	A	100% sama	2147	12.66	100.00	12.83	100.00
2		80% sama	1686	11.47	68.06	11.84	68.48
3		60% sama	1196	10.45	39.38	11.17	39.63
4		40% sama	736	8.95	18.48	10.45	18.73
5		20% sama	264	7.81	2.68	7.95	2.68
6		Tukar 20%	2159	12.38	97.99	13.66	97.99
7		Tukar 40%	2159	12.49	91.97	14.00	92.22
8		Ganti 10%	2081	11.70	96.57	12.53	96.57
9	B	100% sama	1737	7.10	100.00	7.09	100.00
10		80% sama	1391	6.37	70.72	5.99	71.02
11		60% sama	994	5.38	42.55	6.23	42.76
12		40% sama	661	4.72	23.00	4.75	23.30
13		20% sama	297	3.72	7.50	3.83	7.50
14		Tukar 20%	1750	7.09	96.05	7.14	96.15
15		Tukar 40%	1750	7.04	90.37	7.81	90.88
16		Ganti 10%	1735	7.70	92.00	7.40	92.10

17	C	100% sama	4705	30.34	100.00	30.29	100.00
18		80% sama	3670	27.85	68.68	17.06	77.33
19		60% sama	2676	24.76	44.56	23.90	44.88
20		40% sama	1643	20.70	21.50	20.00	21.74
21		20% sama	689	15.36	6.38	14.86	6.41
22		Tukar 20%	4753	32.22	95.37	30.64	95.51
23		Tukar 40%	4753	32.24	90.21	30.85	90.42
24		Ganti 10%	4681	41.66	97.63	38.87	97.80
25	D	100% sama	13192	79.06	100.00	48.93	100.00
26		80% sama	10385	70.02	72.62	65.26	73.12
27		60% sama	7725	61.44	49.02	55.73	49.64
28		40% sama	4954	48.59	28.20	43.91	28.63
29		20% sama	2299	34.26	10.17	35.64	10.50
30		Tukar 20%	13311	80.06	95.28	76.67	95.65
31		Tukar 40%	13311	94.74	91.51	97.14	91.97
32		Ganti 10%	13193	107.95	98.75	99.33	98.83

Lampiran V: Perhitungan manual

Tabel Perhitungan *Hashing* dan Pencocokan *String* Algoritma Rabin-Karp

Dokumen latih A.20-kata.txt

No	Pattern	Nilai Hash	Status
1	plag	47	Found
2	lagi	59	Found
3	agia	74	Found
4	giar	50	Found
5	iari	98	Found
6	aris	81	Found
7	rism	14	Found
8	isme	26	Found
9	smep	65	Found
10	mepe	30	Found
11	epen	99	Found
12	penj	86	Found
13	enji	45	Found
14	njip	57	Found
15	jipl	63	Found
16	ipla	15	Found
17	plak	51	Found
18	laka	91	Found
19	akan	3	Found
20	kanm	42	Found
21	anme	10	Found
22	nmel	10	Found
23	mela	87	Found

No	Pattern	Nilai Hash	Status
24	elan	63	Found
25	lang	26	Found
26	angg	53	Found
27	ngga	25	Found
28	ggar	52	Found
29	garh	16	Found
30	arha	53	Found
31	rhak	35	Found
32	hakp	45	Not Found
33	akpe	43	Not Found
34	kpel	37	Not Found
35	pela	57	Found
36	elak	60	Found
37	laku	10	Found
38	akud	92	Not Found
39	kudi	19	Not Found
40	udis	97	Not Found
41	dise	45	Found
42	iseb	44	Found
43	sebu	48	Found
44	ebut	77	Found
45	butp	74	Found
46	utpl	43	Found
47	tpla	6	Found
48	plag	47	Found
49	lagi	59	Found
50	agia	74	Found
51	giat	52	Found
52	iato	23	Found
53	ator	37	Found

No	Pattern	Nilai Hash	Status
54	torp	82	Found
55	orpl	4	Found
56	rpla	26	Found
57	plag	47	Found
58	lagi	59	Found
59	agia	74	Found
60	giat	52	Found
61	iato	23	Found
62	ator	37	Found
63	tord	70	Found
64	orda	75	Not Found
65	rdap	44	Not Found
66	dapa	19	Not Found
67	apat	4	Not Found
68	patd	43	Not Found
69	atdi	19	Found
70	tdih	96	Not Found
71	dihu	52	Found
72	ihuk	22	Found
73	huku	30	Found
74	ukum	2	Found

Dokumen latih A.40-kata.txt

No	Pattern	Nilai Hash	Status
1	penj	86	Found
2	enji	45	Found
3	njip	57	Found
4	jipl	63	Found
5	ipla	15	Found
6	plak	51	Found

No	Pattern	Nilai Hash	Status
7	laka	91	Found
8	akan	3	Found
9	kanh	37	Not Found
10	anha	57	Not Found
11	nhak	75	Not Found
12	hakc	32	Found
13	akci	18	Found
14	kcip	94	Found
15	cipt	40	Found
16	ipta	95	Found
17	ptap	48	Found
18	tape	65	Found
19	apel	36	Found
20	pela	57	Found
21	elak	60	Found
22	laku	10	Found
23	akud	92	Not Found
24	kudi	19	Not Found
25	udih	86	Not Found
26	dihu	52	Found
27	ihuk	22	Found
28	huku	30	Found
29	ukum	2	Found

Dokumen latih A.60.kata.txt

No	Pattern	Nilai Hash	Status
1	penj	86	Found
2	enji	45	Found
3	njip	57	Found

No	Pattern	Nilai Hash	Status
4	jipl	63	Found
5	ipla	15	Found
6	plak	51	Found
7	laka	91	Found
8	akan	3	Found
9	kanh	37	Not Found
10	anha	57	Not Found
11	nhak	75	Not Found
12	hakc	32	Found
13	akci	18	Found
14	kcip	94	Found
15	cipt	40	Found
16	ipta	95	Found
17	ptap	48	Found
18	tape	65	Found
19	apel	36	Found
20	pela	57	Found
21	elak	60	Found
22	laku	10	Found
23	akud	92	Not Found
24	kudi	19	Not Found
25	udih	86	Not Found
26	dihu	52	Found
27	ihuk	22	Found
28	huku	30	Found
29	ukum	2	Found

Dokumen latih A.80-kata.txt

No	Pattern	Nilai Hash	Status
1	hakp	45	Not Found
2	akpe	43	Not Found
3	kpel	37	Not Found
4	pela	57	Found
5	elak	60	Found
6	laku	10	Found
7	akud	92	Not Found
8	kudi	19	Not Found
9	udih	86	Not Found
10	dihu	52	Found
11	ihuk	22	Found
12	huku	30	Found
13	ukum	2	Found

Dokumen latih AK.20-kal.txt

No	Pattern	Nilai Hash	Status
1	plag	47	Found
2	lagi	59	Found
3	agia	74	Found
4	giar	50	Found
5	iari	98	Found
6	aris	81	Found
7	rism	14	Found
8	isme	26	Found
9	smep	65	Found
10	mepe	30	Found
11	epen	99	Found
12	penj	86	Found
13	enji	45	Found

No	Pattern	Nilai Hash	Status
14	njip	57	Found
15	jipl	63	Found
16	ipla	15	Found
17	plak	51	Found
18	laka	91	Found
19	akan	3	Found
20	kanm	42	Found
21	anme	10	Found
22	nmel	10	Found
23	mela	87	Found
24	elan	63	Found
25	lang	26	Found
26	angg	53	Found
27	ngga	25	Found
28	ggar	52	Found
29	garh	16	Found
30	arha	53	Found
31	rhak	35	Found
32	hakc	32	Found
33	akci	18	Found
34	kcip	94	Found
35	cipt	40	Found
36	ipta	95	Found
37	ptap	48	Found
38	tape	65	Found
39	apel	36	Found
40	pela	57	Found
41	elak	60	Found
42	laku	10	Found
43	akup	3	Found

No	Pattern	Nilai Hash	Status
44	kupl	41	Found
45	upla	97	Found
46	plag	47	Found
47	lagi	59	Found
48	agia	74	Found
49	giat	52	Found
50	iatd	12	Found
51	atdi	19	Found
52	tdis	6	Found
53	dise	45	Found
54	iseb	44	Found
55	sebu	48	Found
56	ebut	77	Found
57	butp	74	Found
58	utpl	43	Found
59	tpla	6	Found
60	plag	47	Found
61	lagi	59	Found
62	agia	74	Found
63	giat	52	Found
64	iato	23	Found
65	ator	37	Found
66	torp	82	Found
67	orpl	4	Found
68	rpla	26	Found
69	plag	47	Found
70	lagi	59	Found
71	agia	74	Found
72	giat	52	Found
73	iato	23	Found

No	Pattern	Nilai Hash	Status
74	ator	37	Found
75	tord	70	Found
76	ordi	83	Found
77	rdih	15	Found
78	dihu	52	Found
79	ihuk	22	Found
80	huku	30	Found
81	ukum	2	Found
82	kumb	1	Found
83	umbe	4	Found
84	mber	37	Found
85	bera	55	Found
86	erat	63	Found

Dokumen latih ZK.40-kal.txt

No	Pattern	Nilai Hash	Status
1	plag	47	Found
2	lagi	59	Found
3	agia	74	Found
4	giar	50	Found
5	iari	98	Found
6	aris	81	Found
7	rism	14	Found
8	isme	26	Found
9	smep	65	Found
10	mepe	30	Found
11	epen	99	Found
12	penj	86	Found
13	enji	45	Found
14	njip	57	Found

No	Pattern	Nilai Hash	Status
15	jipl	63	Found
16	ipla	15	Found
17	plak	51	Found
18	laka	91	Found
19	akan	3	Found
20	kanm	42	Found
21	anme	10	Found
22	nmel	10	Found
23	mela	87	Found
24	elan	63	Found
25	lang	26	Found
26	angg	53	Found
27	ngga	25	Found
28	ggar	52	Found
29	garh	16	Found
30	arha	53	Found
31	rhak	35	Found
32	hakc	32	Found
33	akci	18	Found
34	kcip	94	Found
35	cipt	40	Found
36	ipta	95	Found
37	ptap	48	Found
38	tapl	72	Not Found
39	apla	95	Not Found
40	plag	47	Found
41	lagi	59	Found
42	agia	74	Found
43	giat	52	Found
44	iato	23	Found

No	Pattern	Nilai Hash	Status
45	ator	37	Found
46	tord	70	Found
47	ordi	83	Found
48	rdih	15	Found
49	dihu	52	Found
50	ihuk	22	Found
51	huku	30	Found
52	ukum	2	Found
53	kumb	1	Found
54	umbe	4	Found
55	mber	37	Found
56	bera	55	Found
57	erat	63	Found
58	ratp	35	Not Found
59	atpe	34	Not Found
60	tpel	48	Not Found
61	pela	57	Found
62	elak	60	Found
63	laku	10	Found
64	akup	3	Found
65	kupl	41	Found
66	upla	97	Found
67	plag	47	Found
68	lagi	59	Found
69	agia	74	Found
70	giat	52	Found
71	iatd	12	Found
72	atdi	19	Found
73	tdis	6	Found
74	dise	45	Found

No	Pattern	Nilai Hash	Status
75	iseb	44	Found
76	sebu	48	Found
77	ebut	77	Found
78	buts	77	Not Found
79	utse	66	Not Found
80	tseb	35	Not Found
81	seba	28	Not Found
82	ebag	66	Not Found
83	baga	50	Not Found
84	agai	2	Not Found
85	gaip	35	Not Found
86	aipl	52	Not Found
87	ipla	15	Found
88	plag	47	Found
89	lagi	59	Found
90	agia	74	Found
91	giat	52	Found
92	iato	23	Found
93	ator	37	Found

Dokumen latih AP.10-pa.txt

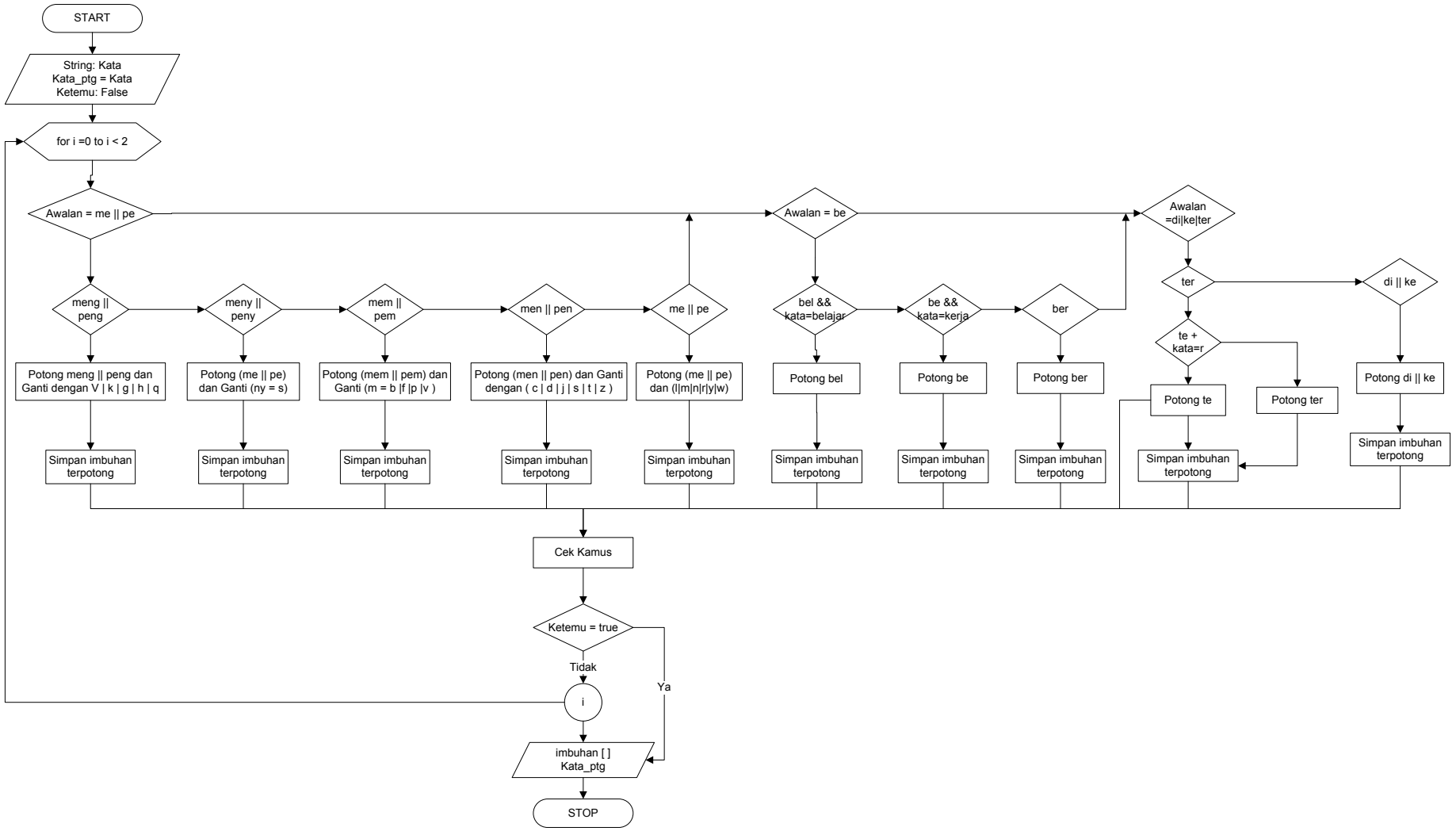
No	Pattern	Nilai Hash	Status
1	plag	47	Found
2	lagi	59	Found
3	agia	74	Found
4	giar	50	Found
5	iari	98	Found
6	aris	81	Found
7	rism	14	Found
8	isme	26	Found

No	Pattern	Nilai Hash	Status
9	smep	65	Found
10	mepe	30	Found
11	epen	99	Found
12	penj	86	Found
13	enji	45	Found
14	njip	57	Found
15	jipl	63	Found
16	ipla	15	Found
17	plak	51	Found
18	laka	91	Found
19	akan	3	Found
20	kanm	42	Found
21	anme	10	Found
22	nmel	10	Found
23	mela	87	Found
24	elan	63	Found
25	lang	26	Found
26	angg	53	Found
27	ngga	25	Found
28	ggar	52	Found
29	garh	16	Found
30	arha	53	Found
31	rhak	35	Found
32	hakc	32	Found
33	akci	18	Found
34	kcip	94	Found
35	cipt	40	Found
36	ipta	95	Found
37	ptap	48	Found
38	tapl	72	Not Found

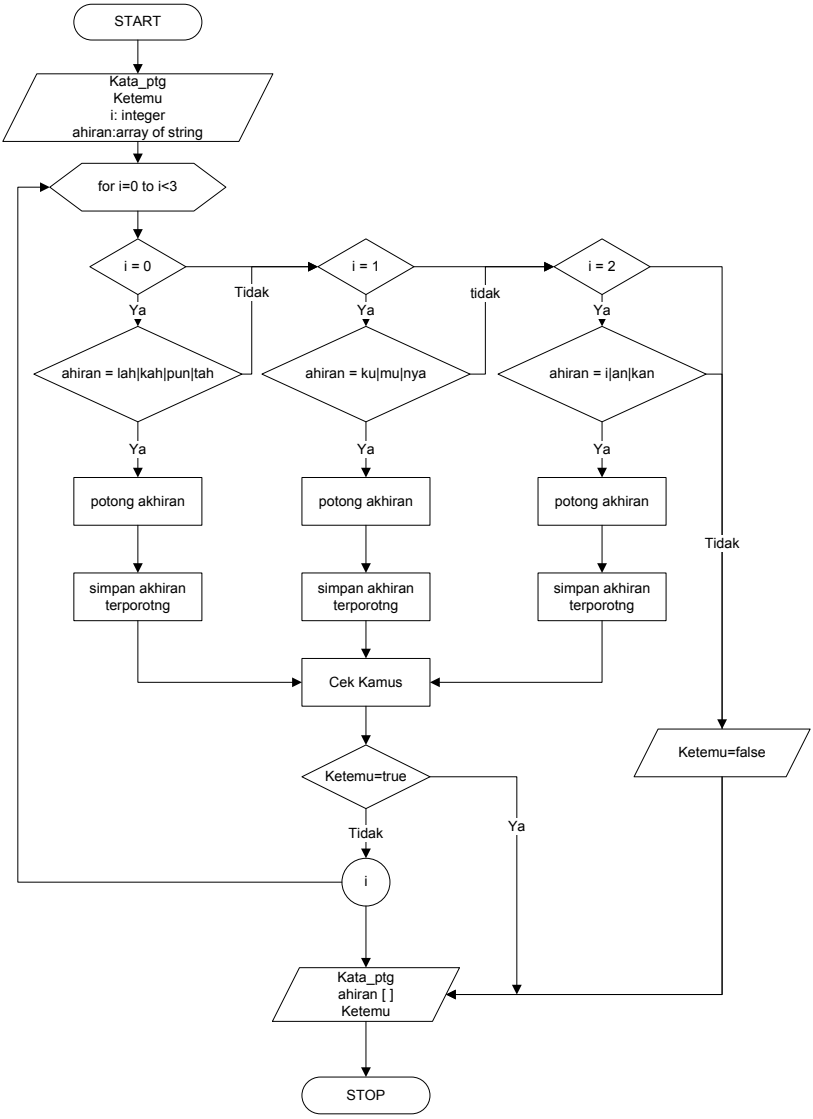
No	Pattern	Nilai Hash	Status
39	apla	95	Not Found
40	plag	47	Found
41	lagi	59	Found
42	agia	74	Found
43	giat	52	Found
44	iato	23	Found
45	ator	37	Found
46	tord	70	Found
47	ordi	83	Found
48	rdih	15	Found
49	dihu	52	Found
50	ihuk	22	Found
51	huku	30	Found
52	ukum	2	Found
53	kumb	1	Found
54	umbe	4	Found
55	mber	37	Found
56	bera	55	Found
57	erat	63	Found
58	ratp	35	Not Found
59	atpe	34	Not Found
60	tpel	48	Not Found
61	pela	57	Found
62	elak	60	Found
63	laku	10	Found
64	akup	3	Found
65	kupl	41	Found
66	upla	97	Found
67	plag	47	Found
68	lagi	59	Found

No	Pattern	Nilai Hash	Status
69	agia	74	Found
70	giat	52	Found
71	iatd	12	Found
72	atdi	19	Found
73	tdis	6	Found
74	dise	45	Found
75	iseb	44	Found
76	sebu	48	Found
77	ebut	77	Found
78	buts	77	Not Found
79	utse	66	Not Found
80	tseb	35	Not Found
81	seba	28	Not Found
82	ebag	66	Not Found
83	baga	50	Not Found
84	agai	2	Not Found
85	gaip	35	Not Found
86	aipl	52	Not Found
87	ipla	15	Found
88	plag	47	Found
89	lagi	59	Found
90	agia	74	Found
91	giat	52	Found
92	iato	23	Found
93	ator	37	Found

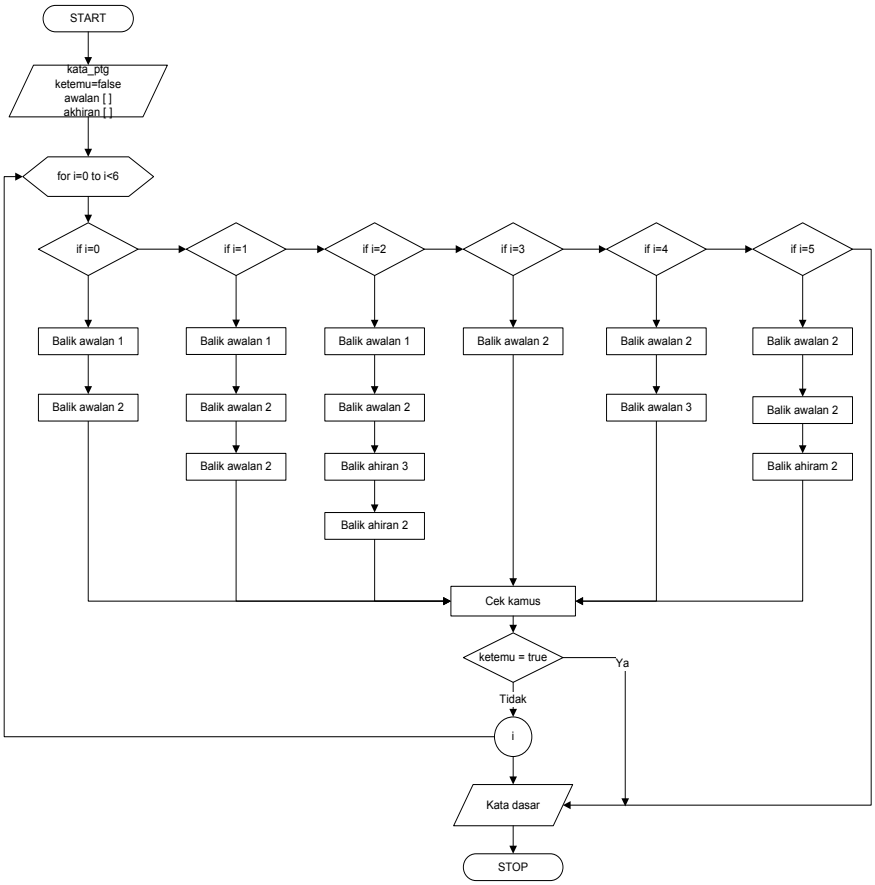
Lampiran VI: Flowchart Stemming



Gambar 5.1 *Flowchart* potong imbuhan



Gambar 5.2 *Flowchart stemming* Arifin – Potong imbuhan



Gambar 5.3 *Flowchart Stemming* Arifin – Cek Kombinasi

Lampiran VII: source code *stemming*

Stemming Arifin

```
1
2       $sudah = false;
3       $pjg_kata = strlen($kata);
4       $awalan1 = array('ber','ter','per');
        $awalan2 =
5   array('me','di','ke','pe','se','be');
6       $vokal   = array('a','i','u','e','o');
7       $akhiran = array('i','nya','kan','an');
8
9       $pos12 = $kata[0].$kata[1];
10      $pos13 = $kata[0].$kata[1].$kata[2];
11      $pos14 = $kata[0].$kata[1].$kata[2].$kata[3];
        $pos15 =
12  $kata[0].$kata[1].$kata[2].$kata[3].$kata[4];
13
14      $pjg_pos12 = strlen($pos12);
15      $pjg_pos13 = strlen($pos13);
16      $pjg_pos14 = strlen($pos14);
17      $pjg_pos15 = strlen($pos15);
18
        $temp_langkah =3; // menyiapkan var jika
19  kalimat tdk memiliki AWII, AKI, AKII, AKIII
20      if(in_array($kata,$kd)){
21          return $kata;
22          exit;
23      }
24
25      // mengecek kata belakang jika KD+AK
26          $kt_blkg13 = substr($kata,0,-3);
27          $kt_blkg12 = substr($kata,0,-2);
28          $kt_blkg11 = substr($kata,0,-1);
```

```
29
        if(in_array($kt_blkg13,$kd)&&
30  in_array(substr($kata,-3),$akhiran)){
31              return $kt_blkg13;
32              exit;
33          }
        if(in_array($kt_blkg12,$kd)&&
34  in_array(substr($kata,-2),$akhiran)){
35              return $kt_blkg12;
36              exit;
37          }
38          if(in_array($kata,$kd)){
39              return $kata;
40              exit;
41          }
42

    if(in_array($pos13,$awalan1)||in_array($pos12,$awa
43  lan2)){
44
45          $P1 = $pos12;
46
47          if($pos12=='me'){
48              $temp_kd = substr($kata,$pjg_pos12);
49
50              if(in_array($temp_kd,$kd)){
51                  return $temp_kd;
52                  exit();
53
54              }
55
56              else
57                  $langkah =2;
58              if($pos14=='meng'){
```

```
59                  $sudah = true;
60                  $P1 = $pos14;
                    $temp_cekHuruf =
61  $kata[$pjg_pos14];
                    //$temp_kd =
62  substr($kata,$pjg_pos14,$pjg_kata);

    if($temp_cekHuruf=='k'||$temp_cekHuruf=='g'||$temp
63  _cekHuruf=='h'){
                        $temp_kd =
64  substr($kata,$pjg_pos14);
65
66                      if(in_array($temp_kd,$kd)){
67                          return $temp_kd;
68                          exit();
69                      }
70
71                      else
72                          $langkah = 2;
73                  }
74

75  if(in_array($temp_cekHuruf,$vokal)){
76                      $P1 = $pos14;
                        $temp_kd   =
77  substr($kata,$pjg_pos14);
                        $temp_kd_1 =
78  'k'.substr($kata,$pjg_pos14);
79
80                      if(in_array($temp_kd_1,$kd)){
81                          return $temp_kd_1;
82                          exit();
83                      }
84
                        else
85  if(in_array($temp_kd,$kd)){
86                          return $temp_kd;
87                          exit();
```

```
88                      }
89
90                      else{
                            $temp_kd =
    substr($kata,$pjg_pos14);
91  //mengamankan,mengalihkan, mengabari..ga bisa
92                          $langkah = 2;
93                      }
94
95                  }
96
97              } // tutup kurung if "meng-"
98
99              if ($pos14=='meny'){
100                 $sudah = true;
101                 $P1 = $pos14;
                    $temp_kd =
102 's'.substr($kata,$pjg_pos14);
103
104                 if(in_array($temp_kd,$kd)){
105                         return $temp_kd;
106                         exit();
107                 }
108
109                 else
110                     $langkah = 2;
111
112             }// tutup kurung if "meny-"
113
114             if($pos13=='mem'){
115                 //$sudah = true;
116                 $P1 = $pos13;
                    $temp_kd =
117 substr($kata,$pjg_pos13);
```

```
                $temp_cekHuruf =
$kata[$pjg_pos13];

                if($temp_cekHuruf=='b'){
                    $temp_kd =
substr($kata,$pjg_pos13);
                    if(in_array($temp_kd,$kd)){
                        return $temp_kd;
                        exit();
                    }

                    else
                       $langkah = 2;

                }

                else if ($temp_cekHuruf=='f'){
                    $temp_kd =
substr($kata,$pjg_pos13);
                    if(in_array($temp_kd,$kd)){
                        return $temp_kd;
                        exit();
                    }

                    else
                       $langkah = 2;

                }

                else{
                     $temp_kd_1 =
'p'.substr($kata,$pjg_pos13);
                     $temp_kd_2 =
'f'.substr($kata,$pjg_pos13);

```

```
148                      if(in_array($temp_kd_1,$kd)){
149                          return $temp_kd_1;
150                          exit();
151                      }
152
153
                         else
154 if(in_array($temp_kd_2,$kd)){
155                          return $temp_kd_2;
156                          exit();
157                      }
158
159                      else{
                             $temp_kd =
160 substr($kata,$pjg_pos13);
161                          $langkah = 2;
162                      }
163
164                 }
165             }// tutup kurung if "mem-"
166
167             if($pos13=='men'&&!$sudah){
168                 $P1 = $pos13;
                    $temp_kd =
169 substr($kata,$pjg_pos13);
                    $temp_cekHuruf =
170 $kata[$pjg_pos13];
171
    if($temp_cekHuruf=='c'||$temp_cekHuruf=='d'||$temp
172 _cekHuruf=='j'){
                        $temp_kd =
173 substr($kata,$pjg_pos13);
174
175                     if(in_array($temp_kd,$kd)){
176                         return $temp_kd;
```

```
177                         exit();
178                     }
179
180                     else
181                         $langkah = 2;
182                 }
                    else
183 if(in_array($temp_cekHuruf,$vokal)){
                        $temp_kd =
184 substr($kata,$pjg_pos13); //'t'
                        //$temp_kd =
185 substr($kata,$pjg_pos12);
186
187                     if(in_array($temp_kd,$kd)){
188                         return $temp_kd;
189                         exit();
190                     }
191
192                     else
193                         $langkah = 2;
194                 }
195
196
                    else
197 if(in_array($temp_cekHuruf,$vokal)){
                        //$temp_kd =
198 't'.substr($kata,$pjg_pos13);
                        $temp_kd =
199 substr($kata,$pjg_pos12);
200
201                     if(in_array($temp_kd,$kd)){
202                         return $temp_kd;
203                         exit();
204                     }
205
206                     else
```

```
207                         $langkah = 2;
208                 }
209             }//tutup kurung if "men-"
210
211         }// tutup kurung "me-"
212
213         else if ($pos12=='pe'){
214             $P1 = $pos12;
215             $temp_kd = substr($kata,$pjg_pos12);
216             $temp_cekHuruf  = $kata[$pjg_pos12];
217             $temp_cekHuruf2 = $kata[$pjg_pos12+1];
218
219             if(in_array($temp_kd,$kd)){
220                 return $temp_kd;
221                 exit();
222             }
223
224             else
225                 $langkah =2;
226
227             if(!in_array($temp_cekHuruf,$vokal)){
228                 $P1 = $pos13;
                    $temp_kd =
229 substr($kata,$pjg_pos13);
230                     if(in_array($temp_kd,$kd)){
231                         return $temp_kd;
232                         exit();
233                     }
234
235                     else
236                         $langkah = 2;
237             }
238
```

```
239             if($temp_cekHuruf=='m'){
240                 $P1 = $pos12;
241 if(in_array($temp_cekHuruf2,$vokal)){
242                     //echo $kata;
243                     $kata_p = $kata;
                        $kata_p[$pjg_pos12] = 'p';
244 //lebur m -> p
                        $temp_kd   =
245 substr($kata,$pjg_pos12);
                        $temp_kd_1 =
246 substr($kata_p,$pjg_pos12);
247
248                     if(in_array($temp_kd_1,$kd)){
249                         return $temp_kd;
250                         exit();
251                     }
252
                        else
253 if(in_array($temp_kd,$kd)){
254                         return $temp_kd;
255                         exit();
256                     }
257
258                     else
259                     $langkah = 2;
260                 }
261
262                 else{
                      $temp_kd   =
263 substr($kata,$pjg_pos13);
264                   $P1 = $pos13;
265                    if(in_array($temp_kd,$kd)){
266                         return $temp_kd;
267                         exit();
268                     }
```

```
269
270                     else{
271                         //$kata_p1 = $kata;
272                         $kata[$pjg_pos12]=' ';
273                         $P1 = $pos13;
                            $temp_kd =
274 substr(trim($kata),$pjg_pos12);
275 if(in_array($temp_kd,$kd)){
276                             return $temp_kd;
277                             exit();
278                         }
279
280                         else
281                         $langkah = 2;
282                     }
283
284                     $langkah =2;
285                 }
286             }//tutup kurung if "m"
287             if($pos14=='peng'){
288                 $sudah = true;
289                 $P1 = $pos14;
                    $temp_cekHuruf =
290 $kata[$pjg_pos14];
                    //$temp_kd =
291 substr($kata,$pjg_pos14,$pjg_kata);
292
293
294 if(in_array($temp_cekHuruf,$vokal)){
295                     $P1 = $pos14;
                        $temp_kd   =
296 substr($kata,$pjg_pos14);
                        $temp_kd_1 =
297 'k'.substr($kata,$pjg_pos14);
298
```

```
299                     if(in_array($temp_kd_1,$kd)){
300                         return $temp_kd_1;
301                         exit();
302                     }
303
304                     else if(in_array($temp_kd,$kd)){
305                         return $temp_kd;
306                         exit();
307                     }
308
309                     else{
310                         $langkah = 2;
311                     }
312
313                 }
314
315                 if($temp_cekHuruf=='k'||$temp_cekHuruf=='g'||$temp_cekHuruf=='h'){
316                     $temp_kd = substr($kata,$pjg_pos14);
317
318                     if(in_array($temp_kd,$kd)){
319                         return $temp_kd;
320                         exit();
321                     }
322
323                     else
324                         $langkah = 2;
325
326
327                 }
328
```

```
329             } // tutup kurung if "peng-"
330
331             else if ($pos14=='peny'){
332                 $sudah = true;
333                 $P1 = $pos14;
334 's'.substr($kata,$pjg_pos14);
335
336                 if(in_array($temp_kd,$kd)){
337                         return $temp_kd;
338                         exit();
339                     }
340
341                     else
342                         $langkah = 2;
343
344             }// tutup kurung if "peny-"
345
346             else if($pos13=='pen'){
347                 $P1 = $pos13;
348 substr($kata,$pjg_pos13);
349 $kata[$pjg_pos13];
350
351 _cekHuruf=='j'){
352 substr($kata,$pjg_pos13);
353
354                     if(in_array($temp_kd,$kd)){
355                         return $temp_kd;
356                         exit();
357                     }
358
```

```
359                     else
360                         $langkah = 2;
361                 }
362
363
                    else
364 if(in_array($temp_cekHuruf,$vokal)){
                        $temp_kd =
365 't'.substr($kata,$pjg_pos13);
366
367                     if(in_array($temp_kd,$kd)){
368                         return $temp_kd;
369                         exit();
370                     }
371
372                     else
373                         $langkah = 2;
374                 }
375
376
377
378             }//tutup kurung if "pen-"
379
380             else if($pos13=='pel'){
                    $temp_kd =
381 substr($kata,$pjg_pos13);
382
383                     if(in_array($temp_kd,$kd)){
384                         return $temp_kd;
385                         exit();
386                     }
387
388             }
389
```

```
390
391         }// tutup kurung if "pe-"
392
393         else if($pos12=='be'){
394
395             $P1 = $pos12;
396             $temp_cekHuruf = $kata[$pjg_pos12];
397             $temp_kd = substr($kata,$pjg_pos12);
398
399             if(in_array($temp_kd,$kd)){
400                 return $temp_kd;
401                 exit();
402             }
403
404             else
405                 $langkah = 2;
406
407             if ($pos12=='be'){
408                 $P1 = $pos12;
                    $temp_kd =
409 substr($kata,$pjg_pos13);
410
411                 if(in_array($temp_kd,$kd)){
412                     return $temp_kd;
413                     exit();
414                 }
415
416                 else
417                     $langkah = 2;
418
419             }
420             else if($pos13=='bel'){
421                $P1 = $pos13;
```

```
                $temp_kd =
422 substr($kata,$pjg_pos13);
423
424                 if(in_array($temp_kd,$kd)){
425                         return $temp_kd;
426                         exit();
427                 }
428
429                 else
430                   $langkah = 2;
431
432             }
433
434
              else
435 if(($pos13=='ber')&&($temp_cekHuruf=='k')){
436                 $P1 = $pos13;
                    $temp_kd =
437 substr($kata,$pjg_pos13);
438
439                 if(in_array($temp_kd,$kd)){
440                         return $temp_kd;
441                         exit();
442                 }
443
444                 else
445                   $langkah = 2;
446
447             } //tutup kurung "ber-"
448           else{
449                 // khusus kata "bekerja"
450                 $P1 = $pos13;
451                 $kata_be = $kata;
452                 $kata_be[$pjg_pos12-1] = '';
```

```
453
                $temp_kd =
454 substr($kata_be,$pjg_pos12);
455
456             if(in_array($temp_kd,$kd)){
457                     return $temp_kd;
458                     exit();
459             }
460
461             else
462               $langkah = 2;
463
464         }
465       }// tutup kurung "be-"
466
467       if ($pos12=='te'){
468           $P1 = $pos12;
469           $temp_kd = substr($kata,$pjg_pos12);
470           //echo $temp_kd.'<br>';
471
472         if(in_array($temp_kd,$kd)){
473             return $temp_kd;
474             exit();
475           }
476
477         else {
478             $P1 = $pos13;
                $temp_kd =
479 substr($kata,$pjg_pos13);
480
481             if(in_array($temp_kd,$kd)){
482                 return $temp_kd;
483                 exit();
```

```
484                 }
485
486                 else
487                     $langkah = 2;
488
489             }
490
491         }// tutup kurung if "te-"
492
493         else if($pos12=='di'){
494             $P1 = $pos12;
495             $temp_kd = substr($kata,$pjg_pos12);
496
497             if(in_array($temp_kd,$kd)){
498                     return $temp_kd;
499                     exit();
500                 }
501
502                 else
503                     $langkah = 2;
504
505         }// tutup kurunf if "di-"
506
507         else if($pos12=="ke"){
508             $P1 = $pos12;
509             $temp_kd = substr($kata,$pjg_pos12);
510
511             if(in_array($temp_kd,$kd)){
512                     return $temp_kd;
513                     exit();
514                 }
515
```

```
516                 else
517                     $langkah = 2;
518
519         } // tutup kurunf if "ke-"
520
521          else if($pos12=="se"){
522             $P1 = $pos12;
523             $temp_kd = substr($kata,$pjg_pos12);
524
525             if(in_array($temp_kd,$kd)){
526                     return $temp_kd;
527                     exit();
528                 }
529
530                 else
531                     $langkah = 2;
532
533         } // tutup kurunf if "se-"
534 }
535
536 // Awalan 2 / Perkiks 2 (P2)
537 if ($langkah==2){
    //echo 'PREFIKS I: temp: '.$temp_kd.'<br>P1:
538 '.$P1.'<br><br>';
539     $P2='';
540     $temp_kd   = trim($temp_kd);
    $p2_pos13  =
541 $temp_kd[0].$temp_kd[1].$temp_kd[2];
542     $pjg_pos13 = strlen(trim($p2_pos13));
543
544
545     if($p2_pos13=='per'){
546         $P2 = $p2_pos13;
```

```
        $temp_kd =
547 substr($temp_kd,strlen($p2_pos13));
548
549         if(in_array($temp_kd,$kd)){
550                 return $temp_kd;
551                 exit();
552             }
553         else
554             $langkah=3;
555
556     }
557
558     else if($p2_pos13=='ber'){
559         $P2 = $p2_pos13;
        $temp_kd =
560 substr($temp_kd,strlen($p2_pos13));
561
562         if(in_array($temp_kd,$kd)){
563                 return $temp_kd;
564                 exit();
565             }
566         else
567             $langkah=3;
568
569     }
570 } // end langkah 2
571
572 // Akhiran 1
573 if($langkah==3){
    $temp_langkah = 3; // mengantisipasi akhiran
574 yang hanya sampai AK 3
    //echo '<br>PREFIKS II temp:
575 '.$temp_kd.'<br>P2 : '.$P2.'<br><br>';
576     $pjg_temp = strlen($temp_kd);
577     $s1_pos13 = substr($temp_kd,-3);
```

```
    //substr($temp_kd,$pjg_temp-3,$pjg_temp);
578     $s1_pos12 = substr($temp_kd,-2);
579     $pjg_s1_pos13 = strlen($s1_pos13);
580     $pjg_s1_pos12 = strlen($s1_pos12);
581
        //echo $temp_kd = substr($temp_kd,0,-
582 $pjg_s1_pos13);
583
584     if($s1_pos13=='lah'||$s1_pos13=='pun'){
585         $S1 = $s1_pos13;
586         $temp_kd = substr($temp_kd,0,-$pjg_pos13);
587
588         if(in_array($temp_kd,$kd)){
589                 return $temp_kd;
590                 exit();
591             }
592         else
593             $langkah=4;
594
595     }
596 } //end akhiran 1
597
598 //Akhiran II
599 if($langkah==4||$temp_langkah==3){
600     $temp_langkah=3;
        //echo '<br>SUFFIKS I temp:
601 '.$temp_kd.'<br>S1: '.$S1.'<br><br>';
602     $pjg_temp = strlen($temp_kd);
        $s2_pos13 = substr($temp_kd,-3);
603 //substr($temp_kd,$pjg_temp-3,$pjg_temp);
604     $s2_pos12 = substr($temp_kd,-2);
605     $pjg_s2_pos13 = strlen($s2_pos13);
606     $pjg_s2_pos12 = strlen($s2_pos12);
607
```

```
    //echo $temp_kd = substr($temp_kd,0,-
608 $pjg_s2_pos13);
609
610     if($s2_pos13=='nya'){
611         $S2 = $s2_pos13;
            $temp_kd = substr($temp_kd,0,-
612 $pjg_s2_pos13);
613
614         if(in_array($temp_kd,$kd)){
615
616                 return $temp_kd;
617                 exit();
618             }
619         else
620             $langkah=5;
621     }
622
623     if($s2_pos12=='mu'){
624         $S2 = $s2_pos12;
            $temp_kd = substr($temp_kd,0,-
625 $pjg_s2_pos12);
626
627         if(in_array($temp_kd,$kd)){
628
629                 return $temp_kd;
630                 exit();
631             }
632         else
633             $langkah=6;
634     }
635
636 }//end langkah 4
637 //Akhiran III
638 if($langkah==5||$temp_langkah==3){
```

```
639
640     $temp_langkah=3;
641
642     $pjg_temp = strlen($temp_kd);
        $s3_pos13 = trim(substr($temp_kd,-3));
643 //substr($temp_kd,$pjg_temp-3,$pjg_temp);
644     $s3_pos12 = trim(substr($temp_kd,-2));
645     $s3_pos11 = trim(substr($temp_kd,-1));
646     $pjg_s3_pos13 = strlen($s3_pos13);
647     $pjg_s3_pos12 = strlen($s3_pos12);
648     $pjg_s3_pos11 = strlen($s3_pos11);
649
        //echo $temp_kd = substr($temp_kd,0,-
650 $pjg_s2_pos13);
651
652       if($s3_pos11=='i'){
653
654         $S3 = $s3_pos11;
          $temp_kd = substr($temp_kd,0,-
655 $pjg_s3_pos11);
656
657         if(in_array($temp_kd,$kd)){
658
659                 return $temp_kd;
660                 exit();
661             }
662         else
663             $langkah=6;
664
665     }
666       //echo $s3_pos12;
       if($s3_pos12=='an'){
667 //strcmp($s3_pos12,'an')||
668
669         $S3 = $s3_pos12;
```

```
670         $temp_kd_an = $temp_kd;
            $temp_kd    = substr($temp_kd,0,-
671 $pjg_s3_pos12);
672
673         if(in_array($temp_kd,$kd)){
674            return $temp_kd;
675            exit();
676         }
677         else
678             $langkah=6;
679
680     }
681
682     if($s3_pos13=='kan'){
683         //echo '<br>kan';
684         $S3 = $s3_pos13;
            $temp_kd_an = substr($temp_kd_an,0,-
685 $pjg_s3_pos13);
686
687         if(in_array($temp_kd_an,$kd)){
688
689                 return $temp_kd_an;
690                 exit();
691             }
692         else
693             $langkah=6;
694     }
695     if($s3_pos11=='i'){
696
697         $S3 = $s3_pos11;
            $temp_kd = substr($temp_kd,0,-
698 $pjg_s3_pos11);
699
700         if(in_array($temp_kd,$kd)){
```

```
701
702                 return $temp_kd;
703                 exit();
704             }
705         else
706             $langkah=6;
707
708     }
709 }//end langkah 5
710 // Lakukan pengecekan syarat
711
    $syarat_e = $P1.$P2.$temp_kd;           // AW
712 I + AW II + KD
    $syarat_f = $P1.$P2.$temp_kd.$S3;       // AW
713 I + AW II + KD + AK III
    $syarat_g = $P1.$P2.$temp_kd.$S3.$S2;   // AW
714 I + AW II + KD + AK III + AK II
    $syarat_i = $P2.$temp_kd;               // AW
715 II + KD
    $syarat_j = $P2.$temp_kd.$S3;           // AW
716 II + KD + AK III
    $syarat_k = $P2.$temp_kd.$S3.$S2;           //
717 AW II + KD + AK III + AK II
718     //echo $syarat_e;
719
720     if(in_array($syarat_e,$kd)){
721          return $syarat_e;
722           exit();
723     }
724
725     if(in_array($syarat_f,$kd)){
726          return $syarat_f;
727          exit();
728     }
729
730     if(in_array($syarat_g,$kd)){
```

```
731           return $syarat_g;
732           exit();
733      }
734
735      if(in_array($syarat_i,$kd)){
736           return $syarat_i;
737           exit();
738      }
739
740      if(in_array($syarat_j,$kd)){
741           return $syarat_j;
742           exit();
743      }
744
745      if(in_array($syarat_k,$kd)){
746           return $syarat_k;
747           exit();
748      }
749              //echo $temp_kd;
750
751              if($P1=='peng'){
752
753                   $temp_kd_1 = 'k'.$temp_kd;
754                   $temp_kd_2 = $temp_kd;
755                   $temp_kd_2[0] = ' ';
756                   $temp_kd_2 = trim($temp_kd_2);
757                   //echo 'xxx';
758                   if(in_array($temp_kd_1,$kd)){
759                   return $temp_kd_1;
760                   exit();
761                   }
762
```

```
763                     if(in_array($temp_kd_2,$kd)){
764                     return $temp_kd_2;
765                     exit();
766                     }
767
768               }
769
770               if($P1=='pe'){
771                     $temp_kd_1 = $temp_kd;
772                     $temp_kd_1[0] = 'p';
773                     //echo $temp_kd_1;
774                     if(in_array($temp_kd_1,$kd)){
775                     return $temp_kd_1;
776                     exit();
777                     }
778
779               }
780
781                if($P1=='meng'){
782                     $temp_kd_1 = 'k'.$temp_kd;
783
784                     if(in_array($temp_kd_1,$kd)){
785                     return $temp_kd_1;
786                     exit();
787                     }
788               }
789
790                if($P1=='mem'){
791                     $temp_kd_1 = 'm'.$temp_kd;
792
793                     if(in_array($temp_kd_1,$kd)){
794                     return $temp_kd_1;
```

```
795                     exit();
796                     }
797                 }
798
799                  if($P1=='men'){
800                     $temp_kd_1 = 'n'.$temp_kd;
801                     $temp_kd_2 = 't'.$temp_kd;
802
803                     if(in_array($temp_kd_1,$kd)){
804                     return $temp_kd_1;
805                     exit();
806                     }
807
808                     if(in_array($temp_kd_2,$kd)){
809                     return $temp_kd_2;
810                     exit();
811                     }
812                 }
813   return $kata;
```

Lampiran VIII : Dokumen Uji

Dokumen Uji A
Penderita asma yang kekurangan vitamin D akan mengalami pemburukan penyakit asma. Defisiensi vitamin ini diduga akan menghalangi reaksi terhadap pengobatan steroid yang sering dipakai dalam obat pelega dan pengontrol asma. Dalam risetnya, para peneliti di National Jewish Health (NJH) di Denver mengukur tingkat vitamin D dari 50 penderita asma dan menilai fungsi paru-paru, hyper-responsiveness saluran udara, yang lazim terjadi di dalam pengerutan saluran udara, dan reaksi terhadap pengobatan steroid. Ternyata, pasien yang mengalami defisiensi vitamin D memperlihatkan hasil yang buruk dalam pemeriksaan fungsi paru-paru dan hyper-responsiveness saluran udara. Lebih dari itu, pasien yang kadar vitamin D dalam tubuhnya di bawah 30 nanogram/mL kadar hyper-responsiveness saluran udara hampir dua kali lipat, dibandingkan dengan mereka yang memiliki lebih banyak vitamin D di dalam darah mereka. Tingkat vitamin D yang rendah juga berkaitan dengan reaksi yang lebih buruk terhadap pengobatan steroid dan peningkatan produksi sitokin pro-peradangan, TNF-alpha. Hal itu meningkatkan kemungkinan bahwa tingkat vitamin D yang rendah berkaitan dengan peningkatan peradangan saluran udara, kata para peneliti itu. Selain itu, kadar vitamin D meramalkan seberapa baik seseorang akan bereaksi terhadap pengobatan steroid bagi asma. "Itu mungkin terjadi karena vitamin D bertindak sebagai pengubah reaksi steroid dengan cara yang sejalan dengan orang yang menderita asma," kata Dr E Rand Sutherland, dari divisi paru-paru dan perawatan medis kritis di NJH. Peserta paling parah memiliki tingkat vitamin D paling rendah, kata mereka. Asma berkaitan dengan kegemukan, dan kekurangan vitamin D mungkin menjadi faktor yang menghubungkan kedua

kondisi tersebut, kata Sutherland.

"Memulihkan tingkat normal vitamin D pada orang yang menderita asma mungkin membantu meningkatkan asma mereka," kata Sutherland. Namun, tidak diketahui apakah asupan vitamin D akan membantu penderita asma, katanya.

Asupan vitamin D buat orang dewasa yang disarankan ialah 300 IU sampai 600 IU, tergantung pada usia, kata US National Institutes of Health.

Dokumen Uji B

Kegiatan olahraga berfungsi membakar kalori agar kalori berlebihan tidak menjadi lemak dalam tubuh. Latihan teratur dengan intensitas sedang sampai berat berperan penting dalam mencegah berbagai penyakit dan membantu memerangi pengaruh merugikan dari kelebihan berat badan.

Olahraga aerobik, minimal 30 menit setiap hari dan dilakukan secara teratur, telah terbukti dapat membakar lemak dan mengencangkan otot-otot. Itu sebabnya latihan aerobik harus jadi bagian dari strategi untuk memangkas kelebihan berat badan.

Kendati begitu, apabila kita menambah latihan aerobik dengan latihan membangun otot, seperti mengangkat beban, hasilnya akan lebih baik. "Jaringan otot memerlukan kalori lebih banyak," kata Janet Walberg Ranin, PhD, profesor di Exercise Science Program di Virginia Polytechnic Institute, Amerika Serikat.

Jika kita meningkatkan massa otot sambil mengurangi lemak, berarti kita meningkatkan kemampuan pembakaran. "Sampai petang hari, ketika Anda duduk dalam rapat atau membaca, otot-otot Anda yang baru terus bekerja membakar kalori," kata Ranin.

Latihan daya tahan tidak hanya sebatas mengangkat barbel. Jika Anda ingin betul-betul melatih kelompok otot yang lain, cara yang

terbaik adalah pergi ke pusat kebugaran dan meminta pelatih menunjukkan latihan-latihan yang tepat. Di gym juga biasanya terdapat berbagai alat untuk menggunakan otot leher, lengan, dada, atau kaki. Jika sudah tahu, Anda bisa mengerjakan latihan serupa di rumah.

Pilihlah latihan teratur yang cukup intens, sering, dan dalam jangka waktu yang cukup dengan jenis latihan yang tepat secara bertahap. Semakin sering Anda mengerjakannya, makin cepat peningkatan massa otot Anda. Dengan demikian, makin banyak pula lemak yang terbakar.

Dokumen Uji C

Android adalah sistem operasi Mobile Phone berbasiskan Linux. Android bersifat open source yang source codenya diberikan secara gratis bagi para pengembang untuk menciptakan aplikasi mereka agar dapat berjalan di Android.Pada mulanya, Android adalah salah satu produk besutan dari Android Inc., namun Google mengakuisisi Android Inc., dan semua kekayaan intelektual milik Android Inc. diperoleh Google Inc. yang kemudian mengembangkan kembali sistem Android.mengakuisi Android Inc..

Sekedar informasi Android Inc. adalah pendatang baru dalam hal membuat software untuk ponsel yang berada di Palo Alto, California Amerika Serikat. Kemudian dibentuklah Open Handset Alliance, konsorsium yang terdiri dari 34 perusahaan hadware, software, dan telekomunikasi, termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, Nvidia, dll. Open Handset Alliance dibentuk untuk mengembangkan Android yang notabene nya adalah OS OpenSource pertama untuk Mobile Phone.

Pada tanggal 5 November 2007, dirilislah Android versi awal dimana Android bersama Open Handset Alliance menyatakan mendukung pengembangan standar terbuka pada perangkat seluler. Di lain pihak, Google merilis kode–kode Android di bawah lisensi Apache, sebuah lisensi perangkat lunak dan standar terbuka perangkat seluler.

Di dunia ini terdapat dua jenis distributor sistem operasi Android. Pertama yang mendapat dukungan penuh dari Google atau Google Mail Services (GMS) dan kedua adalah yang benar–benar bebas distribusinya tanpa dukungan langsung Google atau dikenal sebagai Open Handset Distribution (OHD).

Para pendiri Android Inc. bekerja pada Google, di antaranya Andy Rubi, Rich Miner, Nick Sears, dan Chris White. Saat itu banyak yang menganggap fungsi Android Inc. hanyalah sebagai perangkat lunak pada telepon seluler. Sejak saat itu muncul rumor bahwa Google hendak memasuki pasar telepon seluler.

Di perusahaan Google, tim yang dipimpin Rubin bertugas mengembangkan program perangkat seluler yang didukung oleh kernel Linux. Hal ini menunjukkan indikasi bahwa Google sedang bersiap menghadapi persaingan dalam pasar telepon seluler. hingga sekarang telah banyak ponsel ber-OS Android yang hadir dipasaran, dimulai dari Google Nexus One, HTC Legend, Sony Ericcson Xperia X10, Samsung Galaxy S dan masih banyak lagi.

Keunggulan Android diantaranya :

1. Keterbukaan, Android menyediakan akses ke fungsi dasar perangkat mobile menggunakan standar panggilan ke API.
2. Penghancuran perbatasan, Anda dapat menggabungkan informasi dari Internet ke dalam telepon, seperti informasi kontak, atau data pada lokasi geografis untuk mendapatkan kesempatan baru.
3. Kesamaan aplikasi, Untuk Android ada perbedaan antara telepon utama aplikasi dan perangkat lunak lain, anda bahkan dapat mengubah program untuk memutar nomor, atau screen saver.
4. Cepat dan mudah, perkembangan Dalam SDK memiliki semua yang anda butuhkan untuk membuat dan menjalankan aplikasi Android, termasuk simulator ini instrumen, dan alat debugging maju.

Google mengibaratkan Android sebagai sebuah tumpukan software. Setiap lapisan dari tumpukan ini menghimpun beberapa program yang mendukung fungsi-fungsi spesifik dari sistem operasi. Berikut ini susunan dari lapisan – lapisan tersebut jika di lihat dari lapisan

dasar hingga lapisan teratas:

a. Linux Kernel, Tumpukan paling bawah pada arsitektur Android ini adalah kernel.

b. Android Runtime, Lapisan setelah Kernel Linux adalah Android Runtime.Android Runtime ini berisi Core Libraries dan Dalvik Virtual Machine. Core Libraries mencakup serangkaian inti library Java, artinya Android menyertakan satu set library-library dasar yang menyediakan sebagian besar fungsi-fungsi yang ada pada library-library dasar bahasa pemrograman Java.

c. Libraries, Bertempat di level yang sama dengan Android Runtime adalah Libraries. Android menyertakan satu set library- library dalam bahasa C/C++ yang digunakan oleh berbagai komponen yang ada pada sistem Android.

d. Application Framework, Lapisan selanjutnya adalah application framework, yang mencakup program untuk mengatur fungsi-fungsi dasar smartphone. Application Framework merupakan serangkaian tool dasar seperti alokasi resource smartphone,
aplikasi telepon, pergantian antar – proses atau program, dan pelacakan lokasi fisik telepon.

e. Application, Di lapisan teratas bertempat pada aplikasi itu sendiri. Di lapisan inilah anda menemukan fungsi-fungsi dasar smartphone seperti menelepon dan mengirim pesan singkat, menjalankan web browser, mengakses daftar kontak,
dan lain-lain. Bagi rata-rata pengguna, lapisan inilah yang paling sering mereka akses. Mereka mengakses fungsi- fungsi dasar tersebut melalui user interface.

Dokumen Uji D
Tutorial kali ini akan membahas bagaimana menciptakan dan mengelola kelas dan pewarisan dalam bahasa Java. Pembahasan mengenai kelas akan membahas juga tentang konstruktor yang berfungsi sebagai sarana menciptakan objek dari suatu kelas dan

tentang metode yang merupakan implementasi behaviour dari suatu kelas. Selain itu dibahas juga beberapa jenis kelas khusus seperti kelas bersarang dan kelas anonim. Anda diharapkan telah menguasai konsep pemrograman berorientasi objek yang telah dibahas dalam tutorial sebelumnya.
Kelas

Kelas memiliki peran penting dalam paradigma pemrograman berorientasi objek. Kelas bagaikan sebuah pabrik yang siap memproduksi objek-objek yang kita butuhkan. Untuk dapat menciptakan sebuah objek, kita harus menciptakan kelasnya. Kelas inilah yang dalam runtime program akan menciptakan objek yang inginkan. Singkatnya kelas adalah cetak biru bagi objek yang ingin kita ciptakan.

Perlu diperhatikan bahwa umumnya tidak ada batasan jumlah objek yang dapat diciptakan melalui sebuah kelas. Namun terkadang ada kelas-kelas yang memang dirancang untuk menciptakan tidak lebih dari satu objek.

Objek-objek yang tercipta dari sebuah kelas disebut instans dari kelas tersebut. Instans ini akan mempunyai variabel dan metode yang disebut variabel instans dan metode instans. Namun ada juga variabel kelas yang berlaku bagi kelas itu sendiri sehingga akan berlaku juga untuk semua instans dari kelas tersebut.

Di Listing 1 Anda dapat melihat kode contoh kelas sederhana.Deklarasi Kelas

Pendeklarasian kelas dapat dimulai dengan pernyataan package. Pernyataan seperti baris pertama dari Listing 1 berarti kelas tersebut adalah bagian dari package com.arih. Hirarki package di sini secara konseptual sama dengan konsep pohon, di mana ada node akar dan bercabang terus oleh node anak. Pernyataan ini tidak harus ada dalam setiap kode program kelas. Hanya bila program Anda memiliki banyak kelas dan ingin disusun menurut hirarki tertentu barulah Anda perlu meletakkan pernyataan seperti ini. Biasanya hirarki package mengikuti hirarki domain vendor. Dalam kasus ini,

arih.com—nama situs saya.

Setelah pernyataan package, kita dapat meletakkan baris-baris pernyataan import. Pernyataan import digunakan untuk mendaftarkan nama lengkap dari kelas-kelas yang kita impor agar di dalam kelas ini nantinya kita tidak lagi perlu menuliskan nama lengkap secara hirarkis sebuah kelas. Contohnya di Listing 1 kita meletakkan baris import java.io.*;, artinya semua kelas yang berada di dalam package java.io dapat kita panggil dalam kelas kita langsung dengan nama kelas, tanpa nama package. Jadi bila kita ingin menggunakan kelas java.io.InputStreamReader, kita dapat langsung menuliskan InputStreamReader tanpa mesti membubuhkan java.io di depannya. Dalam kasus tertentu kemudahan ini dapat terganggu bila ada dua kelas dengan nama yang sama namun dari package berbeda yang keduanya kita impor. Akibatnya penggunaan nama kelas tersebut akan menyebabkan ambiguitas. Pemecahannya adalah dengan menggunakan nama lengkap, atau minimal nama package yang berbeda untuk kedua kelas tersebut.

Baris-baris pernyataan impor ini sebenarnya hanya berfungsi sebagai pembantu agar penulisan kode program lebih mudah dan lebih singkat. Jadi hal ini tidak wajib dilakukan. Selama Anda merasa senang menulis nama lengkap suatu kelas yang biasanya cukup panjang, silahkan tidak menggunakan fasilitas impor. Kalaupun Anda menggunakan fasilitas ini, biasakanlah untuk tetap mengetahui nama lengkap dari kelas-kelas yang Anda gunakan. Dengan begitu Anda tidak akan kesusahan bila ingin melihat dokumentasi API Java untuk kelas tersebut.

Konstruktor

Kita sudah mengetahui bahwa kelas adalah alat untuk menciptakan objek. Sekarang yang menjadi pertanyaan adalah bagaimana cara menciptakan objek menggunakan sebuah kelas. Jawabannya adalah dengan menggunakan sebuah konstruktor.

Apakah sebuah konstruktor itu? Konstruktor adalah bagian dari definisi suatu kelas yang berfungsi menciptakan instans dari kelas tersebut. Konstruktor ini bisa kita buat sendiri, atau bila kita tidak

mendefinisikannya, maka kompiler Java akan membuatkan konstruktor default untuk kelas tersebut pada saat kompilasi. Yang perlu diperhatikan adalah bahwa suatu konstrukor tidak termasuk anggota suatu kelas seperti metode dan variabel dan bahwa konstruktor bisa dibuat lebih dari satu.

Bentuk konstruktor sendiri mirip dengan sebuah metode. Beda yang paling mencolok adalah nama sebuah konstruktor harus sama dengan nama kelas tersebut dan konstruktor tidak memiliki definisi return type seperti sebuah metode. Untuk jelasnya Anda bisa lihat kembali Listing 1, di mana terdapat dua buah konstruktor. Yang satu adalah konstruktor default dan yang lain adalah konstruktor dengan sebuah parameter bertipe int.

Adanya dua buah konstruktor ini termasuk sifat polimorfisme dalam pemrograman berorientasi objek. Sifat ini dapat berlaku bagi konstruktor dan juga metode. Lebih jauh mengenai polimorfisme akan dijelaskan lebih jauh di akhir tutorial ini.

Secara singkat, konstruktor adalah perintah yang akan dilaksanakan saat ada instruksi untuk menciptakan sebuah instans dari kelas tersebut. Bisa saja konstruktor tersebut tidak melakukan apa-apa, yaitu bila konstruktor tersebut kosong. Berikut adalah contoh-contoh penciptaan suatu kelas dengan berbagai jenis konstruktor.

```
SuatuKelas kelasContoh = new SuatuKelas();
SuatuKelasLain kelasContoh2 = new SuatuKelasLain("judul");
BufferedReader reader =
   new BufferedReader(new InputStreamReader(System.in));
new SuatuKelas().sebuahMetode();
```

Pada contoh pertama, kita menciptakan sebuah instans dari kelas SuatuKelas dengan memanggil konstruktor tanpa parameter. Bila dalam pendefinisian kelas SuatuKelas sebenarnya tidak ada pendefinisian konstruktor, perintah seperti ini tetap dapat dipakai, karena kompiler Java akan menyisipkan secara otomatis kosntruktor default bila tidak ada definisi konstruktor dalam sebuah kelas. Konstruktor default sendiri sebenarnya tidak melakukan apa-apa

dalam proses instansiasi objek, karena tidak berisi perintah apapun.

Pada contoh kedua, kita memanggil sebuah konstruktor yang mengandung argumen berupa sebuah parameter bertipe String. Konstruktor seperti ini menjadi sangat berguna bila dalam proses instansiasi sebuah objek kita ingin memasukkan suatu nilai untuk dapat digunakan oleh instans baru tersebut.

Pada contoh ketiga, kita memanggil konstruktor dengan argumen sebuah parameter bertipe InputStreamReader. Yang menarik dari contoh ini adalah kita memasukkan argumen berupa kelas baru yang anonim sehingga terjadi pemanggilan konstruktor lain di dalam argumen konstruktor semula. Dalam konstruktor yang terakhir kita juga memasukkan argumen berupa sebuah objek InputStream yaitu System.in.

Contoh keempat adalah contoh yang lebih jelas mengenai kelas anonim. Pada contoh ini kita menciptakan sebuah instans dari kelas SuatuKelas namun tidak menyimpan referensi ke instans tersebut, sehingga kita hanya bisa menggunakannya sekali, yaitu langsung pada baris tersebut seperti terlihat pada contoh.

Ada hal yang menarik dan perlu diperhatikan mengenai konstruktor dalam kelas-kelas yang merupakan subkelas. Pada saat kita menginstans sebuah objek dari sebuah subkelas, kompiler tidak hanya memanggil konstruktor subkelas tersebut, melainkan memanggil kosntruktor superkelasnya terlebih dahulu, baru kemudian memanggil konstruktor subkelas tersebut. Penjelasan mengenai hal ini adalah bahwa sebuah subkelas adalah superkelas itu sendiri namun dengan tambahan karakterisasi yang lebih detil. Sehingga untuk menginstans sebuah subkelas, pertama-tama kita menginstansiasi sebuah superkelas, kemudian menerapkan karakterisasi dari subkelas kepada instans baru tersebut.

Deklarasi Kelas

Deklarasi kelas terdiri dari beberapa komponen. Contoh deklarasi kelas adalah seperti pada Listing 1, yaitu public class KelasKita implements InterfaceKita1, InterfaceKita2 extends SuperKelas. Dari

semua komponen tersebut, yang harus ada adalah bagian class namakelas, sedang yang lain adalah opsional. Tabel 1 akan menunjukkan diagram komponen deklarasi kelas berikut penjelasan singkat.

Kelas Bersarang

Kelas bersarang adalah deklarasi kelas yang terletak di dalam isi kelas lain. Akibatnya kelas baru ini menjadi anggota dari kelas yang melingkupinya.

Kita dapat menggunakan kelas bersarang untuk menerapkan hubungan antara dua kelas dimana kelas bersarang tersebut hanya ada dalam konteks kelas yang melingkupinya. Contohnya, bila kita membuat kelas Listener untuk event yang khusus oleh sebuah kelas, misalnya tombol atau mouse diklik dalam sebuah kelas yang berupa Applet. Maka kita mendeklarasikan kelas Listener tersebut sebagai anggota dari kelas yang memproduksi event.

Selain contoh sebelumnya, kelas bersarang juga dapat digunakan bila sebuah kelas membutuhkan kelas lain untuk dapat berfungsi. Maka kelas tersebut dideklarasikan sebagai anggota dari kelas yang memiliki kebutuhannya. Misalnya kita ingin membuat kelas yang berupa dialog, dan dialog ini membutuhkan sebuah kelas lain sebagai parent dari dialog ini. Maka kita meletakkan kelas dialog tersebut dalam kelas parentnya.

Kelas Anonim

Kelas anonim adalah kelas yang tidak mempunyai nama. Dalam program artinya kelas ini tidak memiliki referensi yang disimpan di simbol tertentu. Akibatnya kita tidak dapat mengakses kelas tersebut dengan menggunakan cara biasa. Satu-satunya cara mengakses kelas seperti ini adalah langsung pada baris instansiasinya. Contoh kelas seperti ini telah kita lihat dalam contoh mengenai konstruktor.

Kelas anonim dapat kita gunakan bila kita ingin menciptakan sebuah kelas namun hanya butuh menggunakannya sekali dan tidak ingin mengalokasikan memori untuknya karena memang tidak akan digunakan lagi. Kasus seperti ini terjadi misalnya kita ingin

menciptakan sebuah kelas baru sebagai argumen untuk sebuah konstruktor atau metode, dimana kita tidak lagi ingin mengakses kelas tersebut secara langsung. Contohnya adalah kasus ketiga dalam contoh mengenai konstruktor. Di sana kita menciptakan sebuah kelas InputStream untuk argumen konstruktor BufferedReader, namun kita tidak ingin menyimpan referensi ke objek InputStream tersebut, karena penggunaannya oleh BufferedReader juga akan berjalan tanpa perintah eksplisit dari program. Jadi sebenarnya referensi ke objek InputStream tersebut disimpan oleh instans BufferedReader, bukan oleh kelas kita.

Kelas Abstrak

Kelas abstrak adalah kelas yang mengandung konsep abstrak sehingga tidak mungkin mempunyai instans. Misalnya suatu kelas abstrak Buah yang mengandung konsep tentang bagian dari tumbuhan yang dapat dimakan. Namun kita tidak dapat menciptakan sebuah instans dari kelas tersebut karena tidak masuk akal menciptakan suatu Buah. Yang mungkin adalah menciptakan instans dari kelas Jeruk, Apel, atau kelas lain yang sudah mengimplementasikan konsep abstrak dari buah.

Kelas abstrak dapat mengandung metode abstrak, yaitu metode yang tidak memiliki implementasi. Dengan begitu, kelas abstrak dapat menentukan bagaimana konsep abstrak tersebut diimplementasikan oleh subkelas yang akan menggunakannya. Kelas abstrak tidak harus memiliki metode abstrak, namun setiap kelas yang memiliki metode abstrak haruslah menjadi kelas abstrak.

Polimorfisme

Polimorfisme secara bahasa dapat diartikan memiliki banyak bentuk. Konsep ini terdapat dalam bahasa pemrograman seperti konstruktor yang memiliki beberapa bentuk. Selain konstruktor, konsep ini juga berlaku bagi metode. Metode atau konstruktor dapat memiliki banyak bentuk dalam arti memiliki nama yang sama namun dengan argumen yang berbeda atau dengan return type yang berbeda. Contoh polimorfisme untuk konstruktor maupun untuk metode dapat Anda lihat pada Listing 1. Disana terdapat konstruktor-konstruktor dengan nama sama namun dengan argumen yang mengandung parameter-

parameter yang berbeda. Untuk contoh polimorfisme untuk metode ditunjukkan bahwa terdapat metode dengan nama sama namun memiliki argumen dan return type yang berbeda.

Kegunaan dari polimorfisme adalah agar kita dapat mendefinisikan beberapa konstruktor atau metode dengan karakteristik yang berbeda-beda agar nantinya dapat digunakan untuk kasus-kasus yang berbeda. Misalnya kita ingin menciptakan instans dari kelas KelasKita pada Listing 1 tanpa memberikan nilai apapun, namun terkadang kita ingin memberikan sebuah nilai sebagai parameter untuk digunakan oleh instans dari kelas tersebut, maka kita dapat membuat kelas seperti KelasKita tersebut. Begitu juga halnya dengan metode, sehingga kita dapat membuat metode-metode yang memiliki karakteristik yang khusus.

Polimorfisme sebenarnya dapat dihilangkan dengan medefinisikan sebuah konstruktor atau metode yang dapat menangani semua kasus yang mungkin. Namun hal ini akan menyebabkan program Anda lebih rumit dan sulit dimengerti. Sedangkan polimorfisme yang membantu Anda untuk membuat program yang lebih baik dan mudah juga membawa konsekuensi yaitu proses kompilasi yang lebih rumit, dan penurunan kecepatan eksekusi kelas. Namun hal ini tidak perlu menjadi perhatian Anda kecuali Anda memang ingin mendalami proses kompilasi Java.