BAB 2
TINJAUAN PUSTAKA

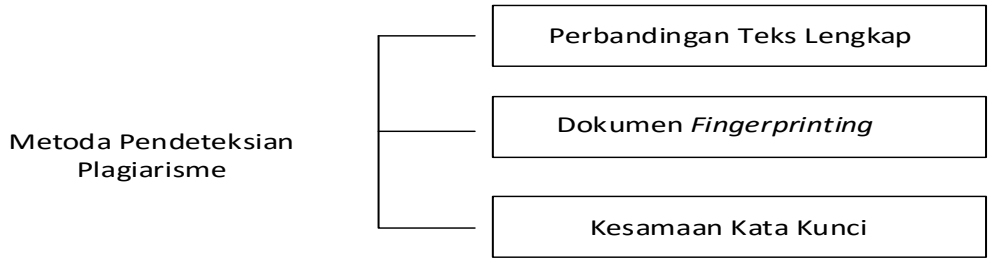
Plagiarisme berasal dari bahasa latin,"*plagiarius*" yang berarti pencuri. *Plagiarisme* didefinisikan sebagai tindakan mengambil, mengumpulkan atau menyampaikan pemikiran, tulisan atau hasil karya orang lain selayaknya itu adalah hasil karya diri sendiri tanpa persetujuan dari pemilik hasil karya tersebut. Dengan kata lain, melakukan plagiarisme bearti mencuri hasil karya atau kepemilikan intelektual orang lain. Ide atau materi apapun yang diambil dari sumber lain untuk penggunaan secara tertulis maupun lisan harus disetujui oleh pemilik hasil karya tersebut, terkecuali informasi tersebut merupakan pengetahuan umum.

Beberapa type plagiarisme yaitu :

1. *Word-to-word plagiarism* adalah menyalin setiap kata secara langsung tanpa diubah sedikit pun.
2. *Plagiarism of authorship* adalah mengakui hasil karya orang lain sebagai karya sendiri dengan cara mencantumkan nama sendiri menggantikan nama pengaran sebenarnya.
3. *Plagiarism of ideas* adalah mengakui hasil pemikiran atau ide orang lain.
4. *Plagiarism of source,* jika seseorang penulis menggunakan kutipan dari penulis lainya tanpa mencantumkan sumbernya. [5]

2.1 Metoda pendeteksian 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 berikut :



Gambar 2. 1 Metoda 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. [6]

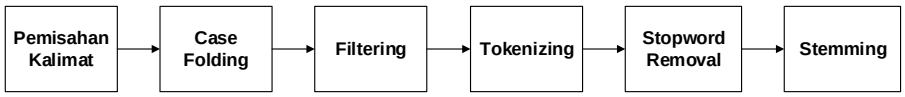
2.2 Pencocokan String (*String Macthing*)

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 bermacammacam, 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.

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.

2.3 Proses *Prepsocessing*

Preprocessing adalah tahapan untuk mempersiapkan teks menjadi data yang akan diolah di tahapan berikutnya. Input-an awal pada proses ini adalah berupa dokumen asli dan dokumen yang akan di uji. Preprocessing pada penelitian ini terdiri dari beberapa tahapan, yaitu: proses pemisahan kalimat, proses *case folding*, proses *filtering*, proses *tokenizing*, proses *stopword* dan proses *stemming*. Berikut gambaran tahap preprocessing dapat dilihat pada gambar 2.2 berikut :



Gambar 2. 2 Gambar tahap *preprocessing*

2.3.1 Pemisahan Kalimat

Pemisahan kalimat adalah proses memecah teks pada dokumen menjadi kumpulan kalimat-kalimat yang merupakan langkah awal tahapan *text preprocessing*. Teknik yang digunakan dalam pemisahan kalimat adalah memisahkan kalimat dengan tanda titik (.), tanda tanya (?), dan tanda seru (!) sebagai pemisah (*delimiter).* Menghilangkan *delimeter* tersebut dokumen akan terpotong menjadi kalimat [7].

Pada proses pemisahan kalimat ini juga akan ada proses pemilihan kalimat , kalimat yang terpadat sumber referensi akan di hapus dan tidak akan diproses ke tahap selanjutnya.

2.3.2 *Casefolding*

Casefolding adalah mengubah semua huruf dalam dokumen menjadi huruf kecil, hanya huruf ‘a’ sampai dengan huruf ‘z’ yang diterima. Karakter selain huruf dihilangkan dan dianggap delimiter. Tahap tokenizing adalah tahap pemotongan string input berdasarkan tiap kata yang menyusunnya [7].

2.3.3 *Filtering*

Data teks dalam dokumen yang sebelumnya sudah diubah ke dalam huruf kecil semua. Selanjutnya dilakukan proses *filtering* teks. *Filtering* adalah tahapan pemrosesan teks dimana semua teks selain karakter “a” sampai “z” akan dihilangkan dan hanya menerima spasi [8].

2.3.4 *Tokenizing*

Proses *Tokenizing* adalah proses pemotongan string input berdasarkan tiap kata yang menyusunnya, memecahkan data kalimat dan memisahkannya menjadi setiap kata. Pemecahan kalimat menjadi kata-kata tunggal dilakukan dengan men-scan kalimat dan setiap kata terindentifikasi atau terpisahkan dengan kata yang lain oleh pemisah spasi [8].

2.3.5 *Stopword Removal*

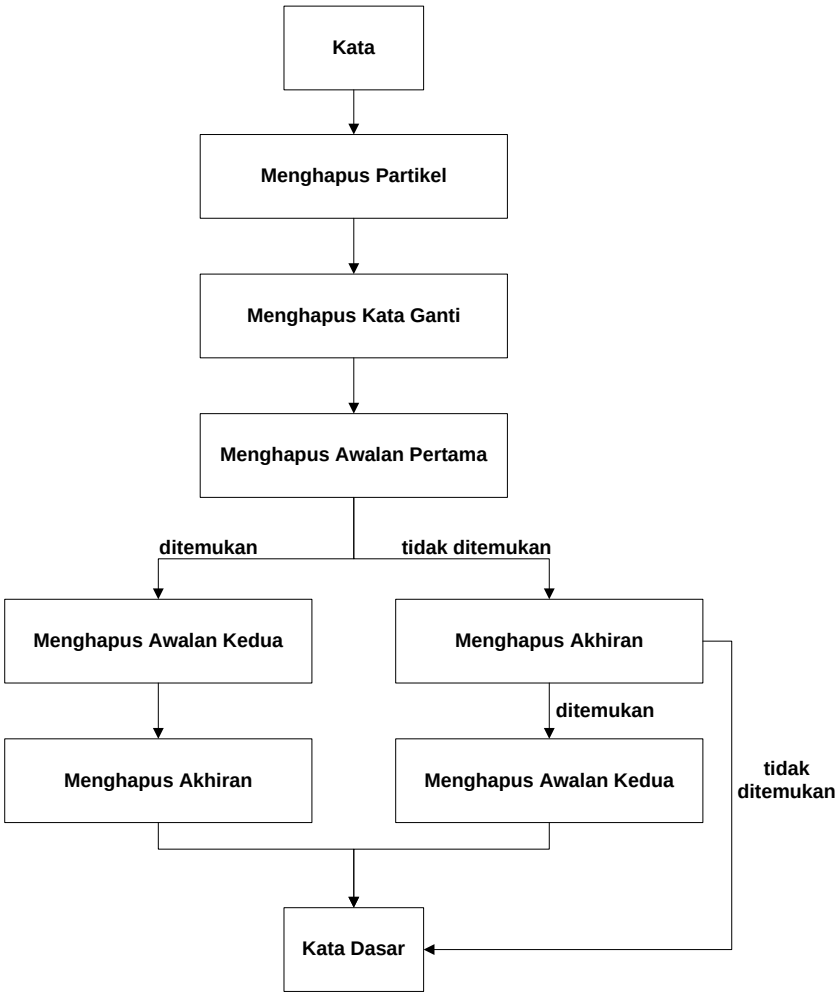
Proses *Stopword Removal* merupakan proses penghilangan *stopword*. Stopword adalah kata-kata yang tidak deskriptif yang dapat dibuang dalam pendekatan bag-of-words [6]. Untuk mendeteksi apakah suatu kata merupakan suatu *stopword* atau bukan adalah menggunakan kamus *stopword* yang sudah ditentukan sebelumnya.

Setiap kata akan diperiksa apakah masuk dalam daftar *stopword*, jika sebuah kata masuk di dalam daftar *stopword* maka kata tersebut tidak akan diproses lebih lanjut dan kata tersebut akan dihilangkan. Sebaliknya apabila sebuah kata tidak termasuk di dalam daftar stopword maka kata tersebut akan masuk keproses berikutnya. [8]. Kandidat umum stopword adalah article, preposisi, dan konjungsi.

2.3.6 *Stemming*

Proses *Stemming* merupakan proses pencarian akar kata (*root word*) dari tiap kata yaitu dengan mengembalikan suatu kata berimbuhan ke bentuk dasarnya (*stem*). Untuk pemrosesan pada bahasa Indonesia, proses *stemming* dilakukan dengan menghilangkan imbuhan yang mengawali dan mengakhiri kata sehingga diperoleh bentuk dasar dari kata tersebut [8]. Algoritma ini dikembangkan oleh Fadillah Z. Tala pada tahun 2003. Berupa algoritma *stemmer* untuk bahasa. Algoritma ini mengadopsi dari algoritma *English Porter Stemmer* yang dikembangkan oleh Frakes pada tahun 1992. Dipilihnya algoritma Porter Stemmer

untuk dikembangkan sebagai algoritma *stemmer* untuk bahasa Indonesia, karena pemikiran dasar dari algoritma Porter Stemmer cocok dengan struktur morfologi kata-kata di dalam bahasa Indonesia[8]. Perbedaan algoritma ini dengan algoritma yang telah dikembangkan oleh Nazief & Adriani yaitu tidak adanya *dictionary* sehingga algoritma ini dapat dikatakan murni berbasis rule[8]. Berdasarkan hasil penelitian [9] Perbedaan algoritma ini dengan algoritma yang telah dikembangkan oleh Nazief & Adriani yaitu tidak adanya *dictionary* sehingga algoritma ini dapat dikatakan murni berbasis rule [9]. Adapun gambar desain dari algoritma Porter Stemmer bahasa Indonesia dapat dilihat pada gambar 2.3 [10].



Gambar 2. 3 Desain Algoritma *Porter Stemmer* Bahasa Indonesia

pada algoritma Porter Stemmer bahasa Indonesia terdapat 5 kelompok aturan yang dapat dilihat pada tabel 2.1 sampai tabel 2.5 berikut [11].

Tabel 2. 1 Kelompok Aturan Pertama : Pembentukan Partikel

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-kah	NULL	2	NULL	bukukah → buku
Akhiran	**Pengganti**	**Kondisi Ukuran**	**Kondisi Tambahan**	**Contoh**
-lah	NULL	2	NULL	pergilah → pergi
-pun	NULL	2	NULL	bukupun → buku
-tah	NULL	2	NULL	apatah → apa

Tabel 2. 2 Kelompok Aturan Kedua : Pembentukan Kata Ganti

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-ku	NULL	2	NULL	bukuku → buku
-mu	NULL	2	NULL	bukumu → buku
-nya	NULL	2	NULL	bukunya → buku

Tabel 2. 3 Kelompok Aturan Ketiga : Pembentukan Awalan Pertama

Awalan Pertama	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
meng-	NULL	2	NULL	mengukur → ukur
meny-	S	2	V…	menyapu → sapu
men-	NULL	2	NULL	menduga → duga
men-	T	2	V…	menuduh → tuduh
mem-	P	2	V…	memilah → pilah
mem-	NULL	2	NULL	membaca → baca
me-	NULL	2	NULL	merusak → rusak
peng-	NULL	2	NULL	pengukur → ukur
peny-	S	2	V…	penyapu → sapu
pen-	NULL	2	NULL	penduga → duga
pen-	T	2	V…	penari → tari

Awalan Pertama	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
pem-	P	2	V…	pemilah → pilah
pem-	NULL	2	NULL	pembaca → baca
di-	NULL	2	NULL	diukur → ukur
ter-	NULL	2	NULL	tersapu → sapu

Tabel 2. 4 Kelompok Aturan Keempat : Pembentukan Awalan Kedua

Awalan Kedua	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
ber-	NULL	2	NULL	berlari → lari
Awalan Kedua	**Pengganti**	**Kondisi Ukuran**	**Kondisi Tambahan**	**Contoh**
bel-	NULL	2	ajar	belajar → ajar
be-	NULL	2	kerja	bekerja → kerja
per-	NULL	2	NULL	perjelas → jelas
pel-	NULL	2	ajar	pelajar → ajar
pe-	NULL	2	NULL	pekerja → kerja

Tabel 2. 5 Kelompok Aturan Kelima : Pembentukan Akhiran

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-kan	NULL	2	Awalan bukan anggota {ke, peng}	tarikkan → tarik
-an	NULL	2	Awalan bukan anggota {di, meng, ter}	perjanjian → janji
-i	NULL	2	Awalan bukan anggota {ber, ke, peng}	mendapati → dapat

2.4 Teknik *Grabbing*

Teknik grabbing adalah teknik mengambil teks atau konten pada situs lain dengan cURL kemudian diletakan pada website lain. Konsep *grabbing* berbeda dari *web service*, dimana informasi atau data yang di ambil tidak disajikan dalam bentuk tertentu sepetti XML atau JSON. Data yang diambil berada pada halaman *website* atau dengan kata lain terselip pada *tag* HTML [12].

Pada teknik *grabbing*, selain membaca *file* HTML dengan cURL diperlukan pula sebuah metoda untuk mengambil data atau informasi yang terselip pada *file* HTML, yaitu dengan *fungsi explode* [13]. Konsep dari *function explode* adalah memotong suatu data dalam hal ini *tag* HTML, berdasar *keyword* tertentu dan menghasilkan data *array*.

Konsep dari teknik *grabbing* sendiri dalam penerapan menggunakan fungsi untuk membaca halaman *website* dan fungsi untuk memotong *tag* HTML, sehingga diperoleh konten atau teks yang dibutuhkan.

2.4.1 Fungsi cURL

cURL (*Client* URL), dikembangkan sebagai alat bantu untuk transfer data dan *file* dengan sintaks URL melalui bermacam – macam protokol. Fungsi – fungsi cURL tersimpan dalam libcurl yang tersedia untuk berbagai bahasa pemrograman termasuk PHP [14] .

cURL digunakan untuk mentransfer data dari *server*. Sebernarnya ada banyak perangkat transfer data layaknya cURL, namum cURL memiliki fitur yang lebih lengkap di antara perangkat – perangkat lainya seperti dukungan terhadap HTTP, FTP, SFTP, SOCKS, TFTP, IMAP, POP3, SMTP dan lain – lain [15] .

2.5 *Synonim Recognition*

Synonym recognition atau pengenalan sinonim adalah salah satu pendekatan yang digunakan untuk membantu proses pendeteksian plagiarisme. Synonym recognition merupakan pendekatan semantik terhadap dokumen teks. Pendekatan ini memanfaatkan kesamaan makna dalam kata yang kemungkinan banyak terjadi. Dengan mendeteksi kata-kata yang sama (sinonim) antara dokumen teks satu dengan yang lain pendekatan ini menambah tingkat keakuratan proses deteksi plagiarism [2]. Algoritma yang digukanan untuk pengenalan dan pemilihan synonim adalah dengan algoritma *lesk.*

2.6 Algoritma *Lesk*

Algoritma lesk adalah algoritma yang digunakan untuk menghilangkan ambiguitas makna kata. Algoritma lesk merupakan salah satu algoritma untuk menyelesaikan masalah word sense disambiguation dengan berbasis kamus. Algoritma ini bekerja dengan membandingkan definisi dari kata yang berambigu

dengan definisI.definisi dari kata tetangganya berdasarkan definisi kamus. Algoritma lesk menggunakan wordnet sebagai kamus atau acuan makna [4].

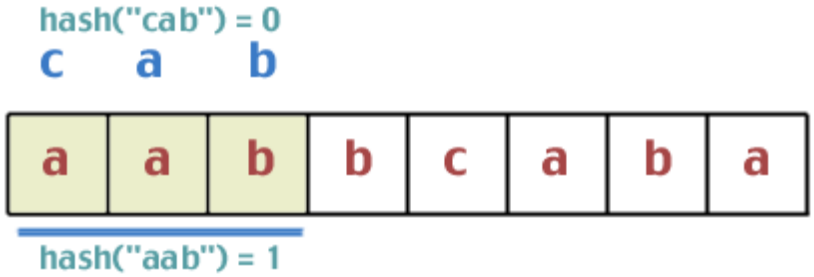
2.7 Metoda *K – Gram*

K-Gram 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-Gram ini digunakan untuk mengambil potongan-potongan karakter huruf sejumlah k dari sebuah kata yang secara kontinuitas dibaca dari teks sumber hingga akhir dari dokumen [12]. Dalam *Markov Model* nilai *K-Gram* yang sering digunakan yaitu, 2-gram (bigram), 3-gram (trigram) dan seterusnya disebut K-Gram (4-gram, 5-gram dan seterusnya). Dalam *natural language processing*, penggunaan K-Gram (atau lebih dikenal dengan ngram), proses parsing token (tokenisasi) lebih sering menggunakan 3-gram dan 4-gram, sedangkan 2-gram digunakan dalam parsing sentence, misal dalam part of-speech (POS). Penggunaan 2-gram dalam tokenisasi akan menyebabkan tingkat perbandingan antar karakter akan semakin besar. Contohnya pada kata „makan‟ dan „mana‟ yang merupakan dua kata yang sama sekali berbeda. Dengan menggunakan metode bigram dalam mencari similarity, hasil dari bigram tersebut yaitu kata “makan” akan menghasilkan bigram ma, ak, ka, an serta kata “mana” akan menghasilkan bigram ma, an, na. Dengan demikian, akan terdapat kesamaan dalam pengecekannya similarity. Namun jika menggunakan 3-gram (“makan” = mak, aka, kan dan “mana” = man,ana) atau 4-gram (“makan” = maka, akan, dan “mana” = mana) akan mengecilkan kemungkinan terjadinya kesamaan pada kata yang strukturnya berbeda [16].

2.8 Algoritma *Rabin Karp*

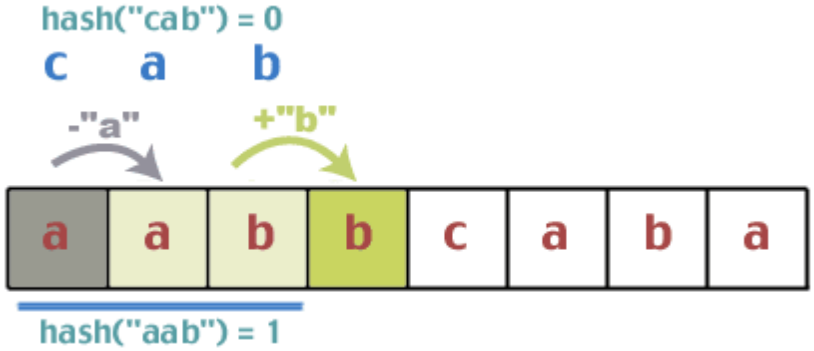
Pada dasarnya, algoritma Rabin-Karp akan membandingkan nilai *hash* dari string masukan dan substring pada teks. Apabila sama, maka akan dilakukan perbandingan sekali lagi terhadap karakter-karakternya. Apabila tidak sama, maka substring akan bergeser ke kanan. Kunci utama performa algoritma ini adala perhitungan yang efisien terhadapa nilai *hash* substring pada saat penggeseran dilakukan. Berikut dijelaskan contoh 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.



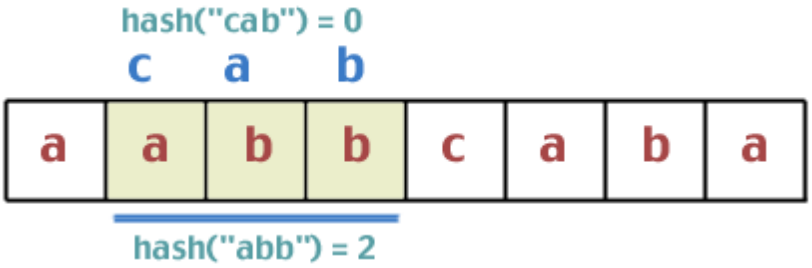
Gambar 2. 4 *Fingerprint* Awal

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.



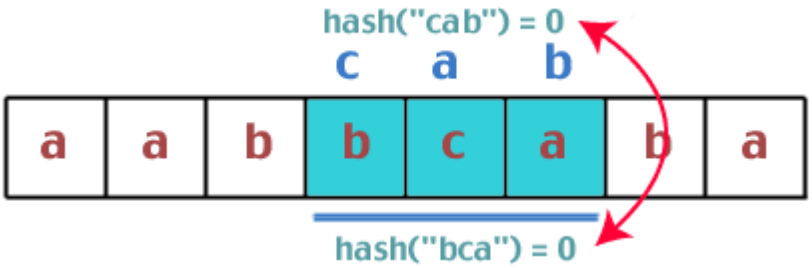
Gambar 2. 5 Menggeser *Fingerprint*

Setelah pergeseran, didapatkan nilai *hash* dari *fingerprint* “abb” (abb = aab – a + b) menjadi dua (2 = 1 –1 + 2).



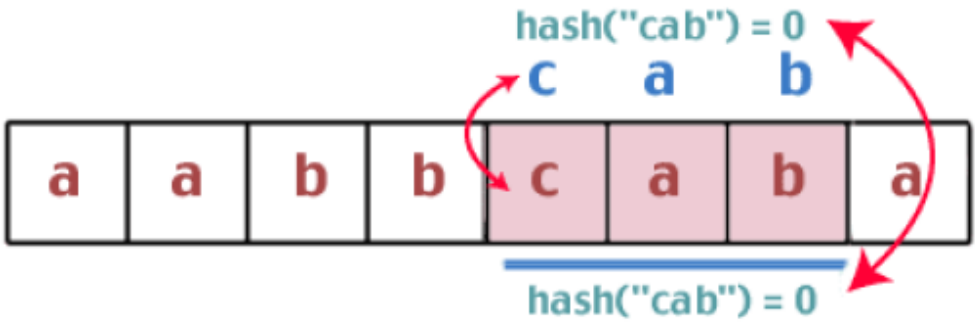
Gambar 2. 6 Pembandingan Kedua

Hasil perbandingan juga tidak sama, maka dilakukan pergeseran. Begitu pula dengan perbandingan ketiga. Pada perbandingan keempat, didapatkan nilai *hash* yang sama.



Gambar 2. 7 Perbandingan keempat (nilai *hash* sama)

Karena nilai *hash* sama, maka dilakukan perbandingan string karakter per karakter antara “bca” dan “cab”. Didapatkan hasil bahwa kedua string tidak sama. Kembali substring bergeser ke kanan.



Gambar 2. 8 Perbandingan kelima (string ditemukan)

Pada perbandingan yang kelima, kedua nilai *hash* dan karakter pembentuk string sesuai, sehingga solusi 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 mangkus daripada kompleksitas waktu yang didapat menggunakan algoritma *brute-force* yaitu *O*(*mn*). [6]

Untuk pola yang panjang dan teks yang besar, algoritma ini menggunakan operasi mod, setelah dikenai operasi mod q, nilainya akan menjadi lebih kecil dari q, missal :

- BAN = 1 + 0 + 13 = 14
  = 14 mod 13 = 1
  = BAN → 1
- CARD = 2 + 0 + 17 + 3 = 22
  = 22 mod 13 = 9
  = CARD → 9

Tetapi tidak semua nilai hash yang cocok berarti polanya cocok. Hal ini sering terjadi pada beberapa kasus, ini disebut spurious hits. Kemungkinan terjadinya diantaranya karena :

- Operasi mod terinterfensi oleh keunikan nilai hash (nilai mod q biasanya dipilih bilangan prima sehingga 10q hanya cocok dengan 1 kata komputer).
  14 mod 13 = 1
  27 mod 13 = 1
- Informasi hilang setelah penjumlahan.
  BAN → 1 + 0 + 13 = 14
  CAM → 2 + 0 + 12 = 14

Sedangkan pseudocode dan rumus matematis yang digunakan adalah sebagai berikut (Cormen, dkk, 2009):

```
RABIN-KARP-MATCHER (T, P, d, q)
n = T.length
m = P.length
h = dm-1 mod q
p = 0
t0 = 0
for i = 1 to m // preprocessing
      p = (dp + P[i]) mod q
      t0 = (dt0 + T[i]) mod q
for s = 0 to n – m
      if p == ts // matching
            if P[1 .. m] == T[s + 1 .. s + m]
             print "Pattern occurs with shift" s
   if s < n – m
      ts+1 = (d(ts - T[s + 1] h) + T[s + m + 1]) mod q
```

Gambar 2. 9 *Pseudocode* Algoritma *Rabin Karp*

Rumus matematis:

t_{s+1} ` = (d (ts – T [s + 1] h) + T [s + m + 1] mod q

Dimana :

ts = Nilai desimal dengan panjang m dari substring T [s + 1 .. s + m], untuk s = 0, 1, ..., n – m.

ts+1 = Nilai desimal selanjutnya yang dihitung dari ts.

d = Radix desimal (bilangan basis 10).

H = *d*m-1

n = Panjang teks

m = Panjang pola

q = Nilai *modulo*

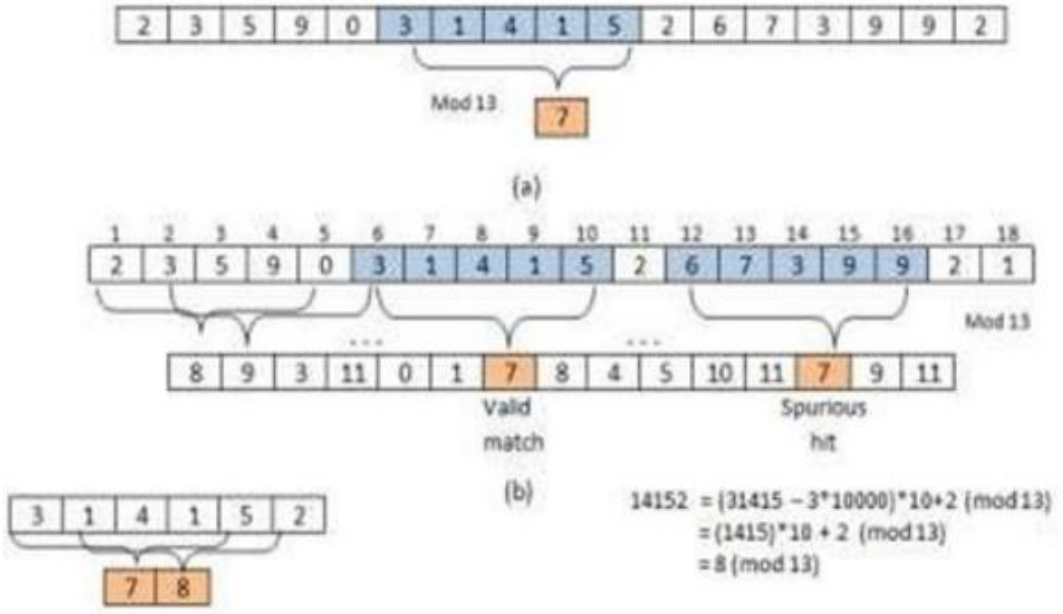
Pengurangan dengan T [s + 1] * h adalah untuk menghilangkan high-order digit dari t s, mengalikan hasilnya dengan 10 untuk menggeser satu digit angka ke kiri, dan menambahkan low-order digit dengan T [s + m + 1]. Misalnya, jika m = 5

dan ts = 31415, maka kita ingin menghapus high-order digit T [s +1] = 3, masukkan low-order digit baru (anggap T [s +5 +1] = 2) dan modulo = 3 untuk memperoleh

ts +1 = (10 (31415 - 3 * 10000)) + 2 mod 13

= 14152 mod 13

= 8



Gambar 2. 10 Gambaran Algoritma *Rabin Karp* Dengan Nilai Mod

2.9 Peningkatan Performa Algoritma Rabin Karp

Telah kita pahami bahwa spurious hit adalah beban tambahan bagi algoritma yang akan meningkatkan waktu proses. Hal ini dikarenakan algoritma harus membandingkan pola terhadap teks yang hasil modulonya sama tetapi nilai hashnya berbeda. Untuk menghindari pencocokan yang tidak perlu, Singh dan Kochar (2008) memberikan solusi untuk tidak hanya membandingkan sisa hasil bagi, tetapi membandingkan hasil baginya juga [17].

$$\text{REM}\ (n1/q) = \text{REM}\ (n2/q) \qquad (2.3)$$

$$\text{QUOTIENT}\ (n1/q) = \text{QUOTIENT}\ (n2/q) \qquad (2.4)$$

Jadi, successful hit harus memenuhi dua syarat, yaitu nilai sisa hasil bagi dan nilai hasil baginya harus sama. Selebihnya adalah unsuccessful hit tanpa perlu

melakukan pencocokan lagi. Hal ini berarti tidak ada pemborosan waktu untuk mengecek spurious hit.

```
RB_IMPROVED (T, P, d, q)
n = T.length
m = P.length
h = dm-1 mod q
p = 0
t0 = 0
Q = 0
pq = 0
h = dm-1 mod q
for i = 1 to m // preprocessing
        p = (dp + P[i]) mod q
       t0 = (dt0 + T[i]) mod q
for s = 0 to n – m
       Q=T{s+1..........s+ m] div q
       If ( ts = p and Q = pq) // matching
                print "Pattern occurs with shift" s
       if s < n – m
                ts+1 = (d(ts - T[s + 1] h) + T[s + m + 1]) mod q
```

Gambar 2. 11 *Pseoudecode Rabin Karp* (2)

2.10 *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. *Hash function* atau fungsi *hash* adalah suatu cara menciptakan *fingerprint* dari berbagai data masukan. *Hash function* akan mengganti atau mentranspose-kan data tersebut untuk menciptakan fingerprint, yang biasa disebut hash value.

Algoritma rabin-karp didasarkan pada fakta jika dua buah *string* sama maka harga *hash* valuenya 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.

Untuk melakukan pengubahan tersebut menggunakan persamaan sebagai berikut :

$$H = C1 * a^{(k-1)} + C2 * a^{(k-2)} + C3 * a^{(k-3)} \ldots + Ck * a0 \quad (2.5)$$

$$\text{Reminder} = H \,/\, \text{Nilai Mod} \quad (2.6)$$

Keterangan:

H = hash value

c = nilai ascii karakter

a = basis

k = banyaknya karakter

2.11 *Dice Similarity Coeficients*

Untuk menghitung nilai similarity dari dokumen fingerprint yang didapat maka digunakan *Dice Similarity Coeficients* dengan cara menghitung nilai dari jumlah K-Gram yang digunakan pada kedua dokumen yang diuji, sedangkan dokumen fingerprint didapat dari jumlah nilai K-Gram yang sama. Nilai Similarity tersebut dapat dihitung dengan menggunakan :

$$S = \frac{2C}{A+B} \quad (2.7)$$

. Dimana S merupakan nilai similarity, dan C merupakan jumlah K-Gram yang sama dari dua buah teks yang di bandingkan, sedangkan A, B merupakan jumlah K-Gram dari masing-masing teks yang dibandingkan. [6]

2.12 Analisis Terstruktur

Model analisis yang digunakan dalam pembangunan aplikasi ini adalah analisis terstruktur. Adapun *tools* pemodelan dalam analisis terstruktur akan dijelaskan di bawah ini.

2.12.1 Diagram Konteks

Diagram Konteks atau sering disebut *Context Diagram* (CD) adalah bagian dari *data flow diagram* yang berfungsi memetakan model lingkungan, yang

dipresentasikan dengan lingkaran tunggal yang mewakili keseluruhan sistem. Dalam CD, sistem masih dianggap kesatuan yang utuh. Entitas luar bisa berupa pengguna, mesin ataupun database yang berada di luar sistem namun berhubungan dengan sistem. Garis masuk dari entitas luar ke dalam sistem menggambarkan input, sedangkan garis keluar dari sistem ke entitas luar menggambarkan output. [18]

2.12.2 Data Flow Diagram

Data Flow Diagram (DFD) adalah suatu model logika data atau proses yang dibuat untuk menggambarkan dari mana asal data dan kemana tujuan data yang keluar dari sistem, dimana data disimpan, proses apa yang menghasilkan data tersebut dan interaksi antara data yang tersimpan dan proses yang dikenakan pada data tersebut. DFD sering digunakan untuk menggambarkan suatu sistem yang telah ada atau sistem baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau dimana data tersebut akan disimpan . [18]

DFD terdiri dari diagram rinci (*DFD Levelled*). DFD levelled menggambarkan sistem sebagai jaringan kerja antara fungsi yang berhubungan satu sama lain dengan aliran dan penyimpanan data, model ini hanya memodelkan sistem dari sudut pandang fungsi. Dalam DFD *levelled* akan terjadi penurunan level dimana dalam penurunan level yang lebih rendah harus mampu merepresentasikan proses tersebut ke dalam spesifikasi proses yang jelas. Jadi dalam DFD *levelled* bisa dimulai dari DFD level 0 kemudian turun ke DFD level 1 dan seterusnya. Setiap penurunan hanya dilakukan bila perlu. Aliran data yang masuk dan keluar pada suatu proses di level x harus berhubungan dengan aliran data yang masuk dan keluar pada level x+1 yang mendefinisikan proses pada level x tersebut.

2.12.3 Spesifikasi Proses

Spesifikasi proses merupakan bagian untuk menjelaskan secara tertulis dari semua proses yang ada dalam DFD. Logika dari proses dituliskan secara baik dan jelas bisa secara deskriptif ataupun pseudo code. [18]

2.12.4 Kamus Data

Kamus Data (KD) adalah katalog fakta tentang data dan kebutuhan-kebutuhan informasi dari suatu sistem [18]. Kamus data merupakan bagian yang berisi dari data yang mengalir pada DFD. Penjelasan struktur data pada tiap data harus sama dengan yang sudah dimodelkan. Tipe data tiap struktur data harus digambarkan dengan sejelas mungkin agar input yang diberikan sesuai. [18]

2.13 Perangkat Lunak Pendukung

Perangkat lunak pendukung merupakan perangkat lunak yang digunakan dalam mendukung pembangunan perangkat lunak. Berikut penjelasan dari perangkat lunak pendukung :

2.13.1 *MySQL*

MySQL adalah sebuah perangkat lunak sistem manajemen basis data SQL atau DBMS yang multithread, multi-user, dengan sekitar 6 juta instalasi di seluruh dunia. MySQL AB membuat MySQL tersedia sebagai perangkat lunak gratis di bawah lisensi GNU General Public License (GPL), tetapi mereka juga menjual dibawah lisensi komersial untuk kasus-kasus dimana penggunaannya tidak cocok dengan penggunaan GPL. Tidak seperti PHP atau Apache yang merupakan *software* yang dikembangkan oleh komunitas umum, dan hak cipta untuk kode sumber dimiliki oleh penulisnya masing-masing, MySQL dimiliki dan disponsori oleh sebuah perusahaan komersial Swedia yaitu MySQL AB. MySQL AB memegang penuh hak cipta hampir atas semua kode sumbernya. Kedua orang Swedia dan satu orang Finlandia yang mendirikan MySQL AB adalah: David Axmark, Allan Larsson, dan Michael "Monty" Widenius [19].

2.13.2 *Xampp*

XAMPP merupakan singakatan dari X (program yang dapat dijalankan diberbagai sistem operasi), Apache, MySQL, PHP, Perl. XAMPP merupakan tool yang menyediakan paket perangkat lunak ke dalam satu buah paket. Terdapat Apache (web server), MySQL (database), PHP (server side scripting), Perl, FTP Server, phpMyAdmin dan berbagai pustaka bantu lainnya. Dengan menginstall XAMPP tidak perlu melakukan instalasi dan konfigurasi web server Apache, PHP

dan MySQL secara manual. XAMPP akan menginstalasi dan mengkofigurasikannya secara otomatis.

2.13.3 PHP

Pembangunan aplikasi untuk menguji akurasi analisis sentimen menggunakan bahasa pemrograman PHP. PHP (singkatan rekrusif PHP : Hypertext Preprocessir) adalah bahasa scripting yang banyak digunakan umum secara open source yang cocok digunakan untuk pengembangan website dengan ditanamkan ke dalam HTML [20]. Kode PHP diawali dengan tanda lebih kecil (<) dan diakhiri dengan tanda lebih besar (>). Ada tiga cara untuk menuliskan script PHP ditampilkan pada gambar 2.14 berikut:

```
<? script php di sini ?>

<?php script php di sini ?>

<script language="php"> script php di sini </script>
```

Gambar 2. 12 Cara Penulisan *Script* PHP

Untuk membuat atau menambahkan komentar, standar penulisannya ditampilkan pada gambar 2.15 berikut:

```
/* komentar */

// komentar

# komentar
```

Gambar 2. 13 Cara Penulisan Komentar PHP