BAB II
LANDASAN TEORI

2.1 Pengertian Plagiarisme

Kata plagiarisme berasal dari kata Latin *plagiarius* yang berarti merampok, membajak. Plagiarisme merupakan tindakan pencurian atau kebohongan intelektual. Plagiarisme adalah tindakan menyerahkan (*submitting*) atau menyajikan (*presenting*) ide atau kata/kalimat orang lain tanpa menyebut sumbernya (Sastroasmoro, 2007).

Plagiarisme dalam Kamus Besar Bahasa Indonesia (KBBI) adalah penjiplakan atau pengambilan karangan (pendapat dan sebagainya) orang lain dan menjadikannya seolah-olah karangan (pendapat dan sebagainya) sendiri (KBBI, 2002). Tindakan plagiat dianggap sebagai tindak pidana karena mencuri hak cipta orang lain. Di dalam dunia akademis, pelaku plagiarisme akan mendapat hukuman berat seperti dikeluarkan dari sekolah/universitas. Pelaku plagiat disebut plagiator (Winoto, 2012).

2.2 Metode Mendeteksi Plagiarisme

Metode pendeteksi plagiarisme dapat di bagi menjadi tiga bagian (Nugroho, 2011), yaitu :

1. Perbandingan Teks Lengkap

   Metode ini di terapkan 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 di simpan 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. Algoritma yang digunakan pada metode ini seperti algoritma *Winnowing*, algoritma *Manber* dan algoritma *Rabin-Karp*. Metode *Rabin-Karp* bisa menelusuri karakter satu persatu pada deret karakter (kontigu) dan proses perbandingannya (perhitungan *hash key*-nya) relatif mudah, tetapi metode ini adalah membutuhkan waktu yang lama dalam membandingkan kata (Taufik, 2012). Algoritma *Manber* proses penyelesaiannya sederhana, dengan waktu yang lebih cepat, dapat dengan mudah di implementasikan, akan tetapi tidak memberikan jaminan bahwa kecocokan antar dokumen terdeteksi. Algoritma *winnowing* memberikan jaminan terdeteksinya kecocokan antar dokumen dan hasilnya lebih informatif karena terdapat informasi posisi *fingerprint* (Kurniawati dan Wicaksana, 2008).

3. Kesamaan Kata Kunci.

   Prinsip dari metode ini adalah mengekstrak kata kunci dari dokumen dan kemudian di bandingkan dengan kata kunci pada dokumen yang lain. Pendekatan yang digunakan pada metode ini adalah teknik dot.

2.3 Information Retrieval

Information Retrieval (IR) adalah menemukan bahan (biasanya dokumen) dari suatu keadaan yang tidak terstruktur (biasanya teks) yang memenuhi kebutuhan informasi dari dalam kumpulan data yang besar (biasanya disimpan didalam komputer)(Manning dkk, 2009). *Information Retrieval* merupakan bagian dari *computer science* yang berhubungan dengan pengambilan informasi dari dokumen-dokumen yang didasarkan pada isi dan konteks dari dokumen-dokumen itu sendiri. Menurut Gerald J. Kowalski (2002) di dalam bukunya "*Information*

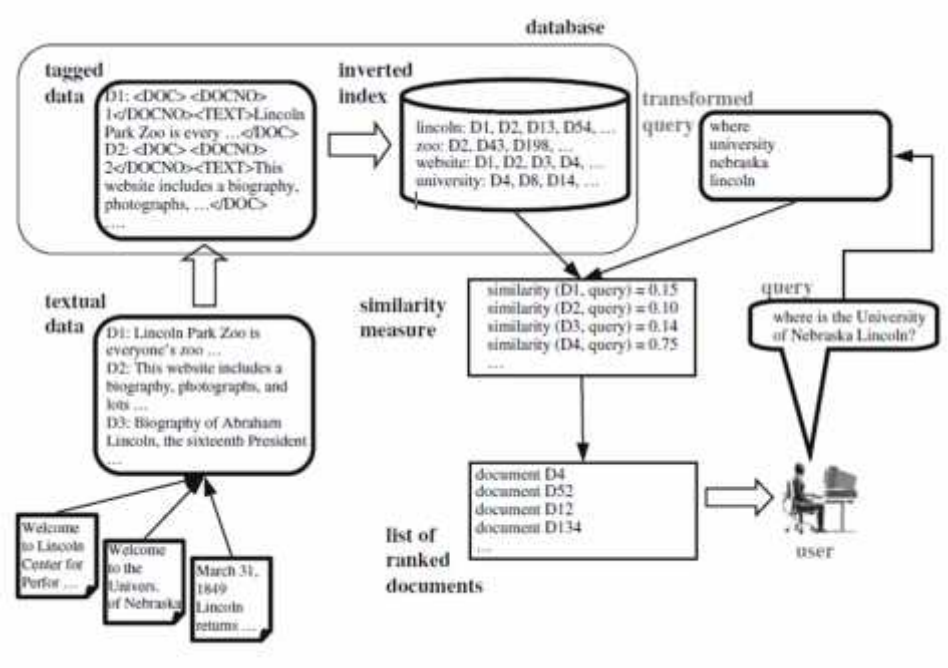
Storage and Retrieval Systems Theory and Implementation", sistem IR adalah suatu sistem yang mampu melakukan penyimpanan, pencarian, dan pemeliharaan informasi. Informasi tersebut dapat berupa teks, gambar, audio, video, dan objek multimedia lainnya.

Tujuan dari sistem IR adalah memenuhi kebutuhan informasi pengguna dengan me-*retrieve* semua dokumen yang mungkin relevan, pada waktu yang sama me-*retrieve* sesedikit mungkin dokumen yang tidak relevan. Sistem IR yang baik memungkinkan pengguna menentukan secara cepat dan akurat apakah isi dari dokumen yang diterima memenuhi kebutuhannya. Agar representasi dokumen lebih baik, dokumen-dokumen dengan topik atau isi yang mirip dikelompokkan bersama-sama (Murad dkk, 2007).

Dalam sistem IR, ada beberapa model yang dapat mereprsentasikan nilai relevan dari dokumen-dokumen, di antaranya model *boolean*, *vector* dan probabalistik. Model *boolean* adalah model yang paling sederhana, berdasarkan teori himpunan dan aljabar *boolean* dan mudah untuk di implementasikan. Tetapi model *Boolean* ini tidak dapat memberikan hasil yang diharapkan dan sangat lambat dalam *run-time* (Jaya, 2007). Model *Vektor* adalah model yang berdasarkan *keyterm*, mendukung *partial matching* dan penentuan peringkat dokumen. Model Probablistik adalah metode yang mengurutkan dokumen dalam urutan menurun terhadap peluang relevansi sebuah dokumen pada informasi yang dibutuhkan (Ramadhany, 2008). Ada beberapa model yang yang dikembangkan berdasarkan perhitungan probablistik yaitu *binary indepence model*, Okapi BM25 dan *Bayessian Network Model.* Model Okapi BM25 telah digunakan secara luas dan berhasil diberbagai koleksi informasi dan tugas pencarian. Terutama dalam evaluasi TREC (*Text Retrieval Conference*), model Okapi BM25 menunjukkan performa yang baik (Manning dkk, 2009). TREC adalah sebuah serial lokakarya yang berfokus pada daftar pencarian informasi (IR) atau trek (Joty dan Al-Hasan, 2007).

2.3.1 Arsitektur Sistem Information Retrieval

Arsitektur sistem IR dapat digambarkan seperti dibawah ini :



Gambar 2.1 Arsitektur dasar sistem IR (Sumber: Cios dkk, 2007)

Ada dua pekerjaan yang ditangani oleh sistem ini, yaitu melakukan *pre-processing* terhadap *database* dan kemudian menerapkan metode tertentu untuk menghitung kedekatan (relevansi) antara dokumen di dalam *database* yang telah di-*preprocess* dengan *query* pengguna (Cios dkk, 2007).

2.3.2 Pembuatan *Index*

Pembuatan *inverted index* harus melibatkan konsep *linguistic processing* yang bertujuan mengekstrak *term-term* penting dari dokumen yang direpresentasikan sebagai *bag-of-words*. Ekstraksi term biasanya melibatkan dua operasi utama berikut (Cios dkk, 2007) :

1. Penghapusan *stop-words*. *Stopword* didefinisikan sebagai *term* yang tidak berhubungan (*irrelevant*) dengan subyek utama dari *database* meskipun kata tersebut sering kali hadir di dalam dokumen. Berikut ini adalah Contoh *stop words* dalam bahasa indonesia : adalah, atau, banyak,

beberapa, dan, demikian, ini, itu, jika, juga, karena, kepada, meskipun, pada, yaitu, yang, ke, di, dia, kami, mereka, saya, dan setiap.

2. *Stemming*. Kata-kata yang muncul di dalam dokumen sering mempunyai banyak varian morfologik. Karena itu, setiap kata yang bukan *stop-words* direduksi ke bentuk *stemmed word* (*term*) yang cocok. Kata tersebut distem untuk mendapatkan bentuk akarnya dengan menghilangkan awalan atau akhiran. Dengan cara ini, diperoleh kelompok kata yang mempunyai makna serupa tetapi berbeda wujud sintaktis satu dengan lainnya. Kelompok tersebut dapat direpresentasikan oleh satu kata tertentu. Sebagai contoh kata "penulisan" dan "tulisan" yang ketika diterapkan stemming akan menghasilkan satu kata dasar yaitu, kata "tulis". Ada beberapa algoritma yang dapat digunakan untuk stemming dalam bahasa indonesia, yaitu algoritma Nazief dan Andriani, algoritma Arifin dan Setiono, algoritma Vega dan algoritma Ahmad, Yussof dan Sembok. Algoritma Nazief dan Andriani adalah algoritma yang paling efektif untuk stemming bahasa Indonesia (Asian dkk, 2004).

Terdapat lima langkah pembangunan *inverted index* (Bintana, 2012), yaitu:

1. Mengumpulkan dokumen yang akan di-*index* (dikenal dengan nama *corpus* atau koleksi dokumen).
2. Penghapusan format dan *markup* dari dalam dokumen.

   Pada tahap ini semua *tag markup* dan format khusus dihapus dari dalam dokumen, terutama pada dokumen yang mempunyai banyak *tag markup* dan format seperti dokumen (X)HTML.
3. Pemisahan rangkaian kata (*tokenization*).

   Pada tahapan ini, seluruh kata di dalam kalimat, paragraf atau halaman dipisahkan menjadi *token* atau potongan kata tunggal atau *termmed word*. Tahapan ini juga akan menghilangkan karakter-karakter tertentu seperti tanda baca dan mengubah semua *token* ke bentuk huruf kecil (*lowercase*).

4. Melakukan *linguistic preprocessing* untuk menghasilkan daftar kata (*token* atau *term*) yang ternormalisasi. Dua hal yang dilakukan dalam tahap ini adalah:
    a. Penyaringan (*filtration*)

       Pada tahapan ini ditentukan *term* mana yang akan digunakan untuk merepresentasikan dokumen sehingga dapat mendeskripsikan isi dokumen dan membedakan dokumen tersebut dari dokumen lain di dalam koleksi. *Term* yang sering dipakai tidak dapat digunakan untuk tujuan ini karena dua alasan. Pertama, jumlah dokumen yang relevan terhadap suatu *query* kemungkinan besar merupakan bagian kecil dari koleksi. *Term* yang efektif dalam pemisahan dokumen yang relevan dari dokumen tidak relevan kemungkinan besar adalah *term* yang muncul pada sedikit dokumen. Ini berarti bahwa *term* dengan frekuensi kemunculan tinggi bersifat *poor descriminator*. Kedua, *term* yang muncul dalam banyak dokumen tidak mencerminkan definisi dari topik atau sub-topik dokumen. Karena itu, *term* yang sering digunakan dianggap sebagai *stop-words* dan dihapus dari dokumen.
    b. Konversi *term* ke bentuk akar (*stemming*)

       Stemming adalah proses konversi *term* ke bentuk akarnya.
5. Mengindeks dokumen (*indexing*).

   Sebuah indeks selalu memetakan kembali dari setiap *term* ke dokumen dimana *term* tersebut muncul. Pengindeksan dilakukan dengan membuat *inverted index* (atau disebut juga *inverted file*) yang terdiri dari *dictionary* dan *postings*. *Inverted index* merupakan konversi dari dokumen asli yang mengandung sekumpulan kata ke dalam daftar kata (*dictionary*) yang berasosiasi dengan dokumen terkait dimana kata-kata tersebut muncul (*postings*). *Dictionary* adalah daftar kata yang diperoleh dari hasil pengindeksan dokumen.

2.3.3 Algoritma Nazief dan Adriani

Algoritma Nazief dan Adriani merupakan algoritma *stemming* untuk teks berbahasa indonesia yang memiliki persentase keakuratan lebih baik dari

algoritma lainnnya. Berikut ini adalah langkah-langkah yang dilakukan oleh algoritma Nazief dan Adriani (Agusta, 2009) :

1. Cari kata yang akan distem dalam kamus. Jika ditemukan maka diasumsikan bahwa kata tesebut adalah *root word*. Maka algoritma berhenti.
2. *Inflection Suffixes* (“-lah”, “-kah”, “-ku”, “-mu”, atau “-nya”) dibuang. Jika berupa *particles* (“-lah”, “-kah”, “-tah” atau “-pun”) maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* (“-ku”, “-mu”, atau “-nya”), jika ada.
3. Hapus *Derivation Suffixes* (“-i”, “-an” atau “-kan”). Jika kata ditemukan di kamus, maka algoritma berhenti. Jika tidak maka ke langkah 3a
    a. Jika “-an” telah dihapus dan huruf terakhir dari kata tersebut adalah “-k”, maka “-k” juga ikut dihapus. Jika kata tersebut ditemukan dalam kamus maka algoritma berhenti. Jika tidak ditemukan maka lakukan langkah 3b.
    b. Akhiran yang dihapus (“-i”, “-an” atau “-kan”) dikembalikan, lanjut ke langkah 4.
4. Hapus *Derivation Prefix*. Jika pada langkah 3 ada sufiks yang dihapus maka pergi ke langkah 4a, jika tidak pergi ke langkah 4b.
    a. Periksa tabel kombinasi awalan-akhiran yang tidak diizinkan (tabel 2.1). Jika ditemukan maka algoritma berhenti, jika tidak pergi ke langkah 4b.
    b. For i = 1 to 3, tentukan tipe awalan kemudian hapus awalan. Jika *root word* belum juga ditemukan lakukan langkah 5, jika sudah maka algoritma berhenti. Catatan: jika awalan kedua sama dengan awalan pertama algoritma berhenti.
5. Melakukan *Recoding*.
6. Jika semua langkah telah selesai tetapi tidak juga berhasil maka kata awal diasumsikan sebagai *root word*. Proses selesai.

Tipe awalan ditentukan melalui langkah-langkah berikut:

1. Jika awalannya adalah: “di-”, “ke-”, atau “se-” maka tipe awalannya secara berturut-turut adalah “di-”, “ke-”, atau “se-”.
2. Jika awalannya adalah “te-”, “me-”, “be-”, atau “pe-” maka dibutuhkan sebuah proses tambahan untuk menentukan tipe awalannya.
3. Jika dua karakter pertama bukan “di-”, “ke-”, “se-”, “te-”, “be-”, “me-”, atau “pe-” maka berhenti.
4. Jika tipe awalan adalah “tidak ada” maka berhenti. Jika tipe awalan adalah bukan “tidak ada” maka awalan dapat dilihat pada Tabel 2.2. Hapus awalan jika ditemukan.

Tabel 2.1 Kombinasi Awalan Akhiran yang Tidak Diizinkan

Awalan	Akhiran yang tidak diizinkan
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
Se-	-i, -kan

Tabel 2.2 Jenis Awalan Kata yang Berawalan Te-

Following Characters				Tipe Awalan
Set 1	Set 2	Set 3	Set 4	
-r-	-r-	-	-	tidak ada
-r-	Vokal	-	-	ter-luluh
-r-	Bukan (-r- atau Vokal)	-er-	vokal	ter
-r-	Bukan (-r- atau vokal)	-er-	Bukan vokal	tidak ada
Bukan (vokal atau –r-)	-er=	Vokal	-	tidak ada
Bukan (vokal atau –r-)	-er-	Bukan vokal	-	te

Tabel 2.3 Jenis Awalan Berdasarkan Tipe Awalnya

Awalan	Awalan yang harus dihapus
di-	di-

Tabel 2.3 Jenis Awalan Berdasarkan Tipe Awalnya (Lanjutan)

Awalan	Awalan yang harus dihapus
ke-	ke-
se-	Se-
te-	te-
ter-	ter-
ter-luluh	ter

2.3.4 Pembobotan Kata

Setiap *term* yang telah di-*index* diberikan bobot sesuai dengan skema pembobotan yang dipilih, apakah pembobotan lokal, global atau kombinasi keduanya. Jika menggunakan pembobotan lokal maka, pembobotan *term* diekspresikan sebagai *tf* (*term frequency*). Namun, jika pembobotan global yang digunakan maka, pembobotan *term* didapatkan melalui nilai *idf* (*inverse document frequency*). Beberapa aplikasi juga ada yang menerapkan pembobotan kombinasi keduanya yaitu, dengan mengalikan bobot lokal dan global (*tf* . *idf*) (Bintana, 2012).

1. *Term Frequency*

   Empat cara yang dapat digunakan untuk memperoleh nilai *term frequency* (*tf*), yaitu:

   a. *Raw term frequency*. Nilai *tf* sebuah *term* diperoleh berdasarkan jumlah kemunculan *term* tersebut dalam dokumen. Contohnya, jika suatu *term* muncul sebanyak tiga kali dalam suatu dokumen maka, nilai *tf term* tersebut adalah 3.

   b. *Logarithm term frequency*. Hal ini untuk menghindari dominasi dokumen yang mengandung sedikit *term* dalam *query*, namun

mempunyai frekuensi yang tinggi. Cara ini menggunakan fungsi logaritmik matematika untuk memperoleh nilai *tf*.

tf=1+ log(tf) .....(2.1)

c. *Binary term frequency*. Hanya memperhatikan apakah suatu *term* ada atau tidak dalam dokumen. Jika ada, maka *tf* diberi nilai 1, jika tidak ada diberi nilai 0. Pada cara ini jumlah kemunculan *term* dalam dokumen tidak berpengaruh.

d. *Augmented term frequency*. Nilai *tf* adalah jumlah kemunculan suatu *term* pada sebuah dokumen, sedangkan nilai *max(tf)* adalah jumlah kemunculan terbanyak sebuah *term* pada dokumen yang sama.

$$tf = 0.5 + 0.5 \times \frac{tf}{\max(tf)} \quad \text{..... (2.2)}$$

2. *Inverse Document Frequency*

Inverse document frequency (*idf*) digunakan untuk memberikan tekanan terhadap dominasi *term* yang sering muncul di berbagai dokumen. Hal ini diperlukan karena *term* yang banyak muncul di berbagai dokumen, dapat dianggap sebagai *term* umum (*common term*) sehingga tidak penting nilainya. Pembobotan akan memperhitungkan faktor kebalikan frekuensi dokumen yang mengandung suatu *term* (*inverse document frequency*).

$$idf\ t = \log\left(\frac{N}{df(t)}\right) \quad \text{..... (2.3)}$$

Keterangan:

N : jumlah dokumen dalam *corpus*.

df_t : *document frequency* atau jumlah dokumen dalam *corpus* yang mengandung *term t*.

2.4 Model Okapi Best Match (BM) 25

Model Okapi BM25 merupakan kombinasi model probabilistik dan pembobotan lokal (*term frequency*). Nilai paling sederhana untuk dokumen *d* adalah hanya dengan menghitung bobot *idf* dari *term* yang terdapat pada *query* yang diinputkan oleh *user* (Bintana, 2012).

$$RSV\ d = \sum_{t \in q} Log\ \frac{N}{df(t)} \quad \text{.....(2.4)}$$

Dari persamaan diatas dapat dikembangkan dengan memfaktorkan frekuensi masing-masing *term* dan panjang dokumen seperti yang terjadi pada model Okapi BM25. Persamaan model Okapi BM25 ditunjukkan pada Persamaan berikut (Bintana, 2012).

$$RSV\ d\ =\ \sum_{t\in q}\log\ \frac{N}{df\ t}\ .\frac{k1+1\ .tf\ td}{k1\ \ 1-b\ +b\times\ \frac{Ld}{Lave}\ \ +tf\ td}.\frac{k3+1\ .tf(tq)}{k3+tf(tq)}\ \text{.....}\ (2.5)$$

Persamaan diatas digunakan jika query yang dimasukkan *user* panjang atau dalam bentuk paragraf yang memungkinkan terdapat *term* yang ganda. untuk *query* yang pendek (tidak terdapat *term* ganda) dapat menggunakan persamaan berikut (Bintana, 2012).

$$RSV\ d\ =\ \sum_{t\in q} log\ \frac{N}{df(t)}\ .\frac{k1+1\ .tf(td)}{k1\ \ 1-b\ +b\times\ \frac{Ld}{Lave}\ \ +tf(td)}\text{.....(2,6)}$$

Keterangan

N : jumlah dokumen dalam *corpus*.

df(t) : jumlah dokumen yang mengandung *term* t pada *query* q.

tf(td) : frekuensi *term* t dalam dokumen d.

tf(tq) : frekuensi *term* t dalam *query* q.

Ld : panjang dokumen d.

Lave : rata-rata panjang dokumen secara keseluruhan.

k1 : konstanta frekuensi *term* dalam dokumen = $1{,}2 \le k1 \le 2$

k3 : konstanta frekuensi *term* dalam *query* = $1{,}2 \le k3 \le 2$

b : konstanta panjang dokumen = $0 \le b \le 1$

2.5 Algoritma Winnowing

Winnowing adalah algoritma yang digunakan untuk melakukan proses *document fingerprinting*. Proses ini ditujukan agar dapat mengidentifikasi penjiplakan, termasuk bagian-bagian kecil yang mirip dalam dokumen yang berjumlah banyak. Input dari proses *document fingerprinting* adalah file teks. Kemudian output-nya akan berupa sekumpulan nilai hash yang disebut *fingerprint*. *Fingerprint* inilah yang akan dijadikan dasar pembanding antar file-file teks yang telah dimasukkan. Algoritma yang digunakan untuk mencari nilai

hash dalam winnowing adalah *rolling hash*. Salah satu prasyarat dari algoritma deteksi penjiplakan adalah *whitespace insensitivity* (tidak terpengaruh oleh spasi, jenis huruf (kapital atau normal), tanda baca dan sebagainya), *Winnowing* telah memenuhi prasyarat tersebut dengan cara membuang seluruh karakter-karakter yang tidak relevan misal: tanda baca, spasi dan juga karakter lain, sehingga nantinya hanya karakter-karakter yang berupa huruf atau angka yang akan diproses lebih lanjut (Kusmawan dkk, 2010).

Secara garis besar, berikut konsep algoritma *Winnowing* bekerja (Pratama dkk, 2012) :

1. Penghapusan karakter-karakter yang tidak relevan (*whitespace insensitivity*). Sehingga hal yang mengandung huruf kapital dijadikan huruf kecil, tanda baca, spasi, dan karakter-karakter yang tidak relevan lainnya dibuang.

   Misalnya kalimat

   "Teknik Informatika adalah jurusan terbaik di UIN SUSKA"

   akan menjadi

   "`teknikinformatikaadalahjurusanterbaikdiuinsuska`".

2. Pembentukan rangkaian gram dengan ukuran k. *K-gram* adalah rangkaian terms dengan panjang k. Kebanyakan yang digunakan sebagai *terms* adalah kata. *K-gram* merupakan sebuah metode yang diaplikasikan untuk pembentukan 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 (Ramadhani dkk, 2012). Misalnya kalimat

   "teknikinformatikaadalahjurusanterbaikdiuinsuska"

   Akan bentuk menjadi rangkaian *gram* dengan ukuran 8-*gram* maka akan menjadi

   ```
   teknikin  eknikinf  knikinfo  nikinfor  ikinform  kinforma
   informat  nformati  formatik  ormatika  rmatikaa  matikaad
   atikaada  tikaadal  ikaadala  kaadalah  aadalahj  adalahju
   dalahjur  alahjuru  lahjurus  ahjurusa  hjurusan  jurusant
   urusante  rusanter  usanterb  santerba  anterbai  nterbaik
   ```

terbaikd erbaikdi rbaikdiu baikdiui aikdiuin ikdiuins kdiuinsu diuinsus iuinsusk uinsuska

3. Penghitungan nilai *hash*. Fungsi yang digunakan untuk menghasilkan nilai *hash* dari rangkaian *gram* dalam algoritma *winnowing* adalah *rolling hash*. $H(_{c1...ck})$ didefinisikan sebagai berikut :

   $H(_{c1...ck})=c_1 * b^{(k-1)} + c_2 * b^{(k-2)} + ... + c_{(k-1)} * b^k + c_k$ ..... (2,7)

   Keterangan:

   c : nilai ascii karakter

   b : basis (bilangan prima)

   k : banyak karakter

   Keuntungan dari *rolling hash* adalah untuk nilai *hash* berikutnya $H(_{c2...ck+1})$ dapat dilakukan dengan cara:

   $H_{(c2...ck+1)} = (H_{(c1...ck)} - c_1 * b^{(k-1)}) * b + c_{(k+1)}$ ..... (2,8)

   Dengan begitu tidak perlu melakukan iterasi dari indeks pertama sampai terakhir untuk menghitung nilai *hash* untuk *gram* ke-2 sampai terakhir. Hal ini tentu dapat menghemat biaya komputasi saat menghitung nilai *hash* dari sebuah *gram* (Kusmawan dkk, 2010).

   Dari rangkaian *gram* pada tahap 2 dihitung *rolling hash* dimana b=11 dan k=8 maka diperoleh nilai *hash* sebagai berikut

   2458436431 2177170647 2298630247 2348532564 2254381403
   2290513025 2259243124 2343991964 2204434801 2384177046
   2432111812 2316317598 2114375646 2465320757 2252898228
   2274198107 2079779016 2084757836 2139524853 2098885400
   2294928058 2093449587 2235134110 2293151702 2502627437
   2448912844 2501128948 2432429448 2105452718 2367168548
   2459377218 2187519307 2412465513 2100208314 2095121226
   2253522144 2281061196 2155273004 2272115051
   2485583153

4. Membagi ke dalam *window* tertentu. Nilai-nilai *hash* yang telah terbentuk, selanjutnya dibentuk dalam beberapa *window* dengan ukuran *w*. *Window*

merupakan pembagian atau pengelompokan beberapa nilai *hash* dengan ukuran yang ditentukan (Ridho, 2013). Misalkan nilai-nilai *hash* pada tahap 3 dibentuk window dengan ukuran w=4 maka diperoleh

{ 2458436431 2177170647 2298630247 2348532564 }
{ 2177170647 2298630247 2348532564 2254381403 }
{ 2298630247 2348532564 2254381403 2290513025 }
{ 2348532564 2254381403 2290513025 2259243124 }
{ 2254381403 2290513025 2259243124 2343991964 }
{ 2290513025 2259243124 2343991964 2204434801 }
{ 2259243124 2343991964 2204434801 2384177046 }
{ 2343991964 2204434801 2384177046 2432111812 }
{ 2204434801 2384177046 2432111812 2316317598 }
{ 2384177046 2432111812 2316317598 2114375646 }
{ 2432111812 2316317598 2114375646 2465320757 }
{ 2316317598 2114375646 2465320757 2252898228 }
{ 2114375646 2465320757 2252898228 2274198107 }
{ 2465320757 2252898228 2274198107 2079779016 }
{ 2252898228 2274198107 2079779016 2084757836 }
{ 2274198107 2079779016 2084757836 2139524853 }
{ 2079779016 2084757836 2139524853 2098885400 }
{ 2084757836 2139524853 2098885400 2294928058 }
{ 2139524853 2098885400 2294928058 2093449587 }
{ 2098885400 2294928058 2093449587 2235134110 }
{ 2294928058 2093449587 2235134110 2293151702 }
{ 2093449587 2235134110 2293151702 2502627437 }
{ 2235134110 2293151702 2502627437 2448912844 }
{ 2293151702 2502627437 2448912844 2501128948 }
{ 2502627437 2448912844 2501128948 2432429448 }
{ 2448912844 2501128948 2432429448 2105452718 }
{ 2501128948 2432429448 2105452718 2367168548 }
{ 2432429448 2105452718 2367168548 2459377218 }
{ 2105452718 2367168548 2459377218 2187519307 }
{ 2367168548 2459377218 2187519307 2412465513 }
{ 2459377218 2187519307 2412465513 2100208314 }
{ 2187519307 2412465513 2100208314 2095121226 }
{ 2412465513 2100208314 2095121226 2253522144 }
{ 2100208314 2095121226 2253522144 2281061196 }
{ 2095121226 2253522144 2281061196 2155273004 }
{ 2253522144 2281061196 2155273004 2272115051 }

```
{ 2281061196 2155273004 2272115051 2485583153 }
```

5. Pemilihan beberapa nilai *hash* menjadi document *fingerprinting*. Dari *window* yang telah dibentuk dilakukan pemilihan nilai *hash* terkecil pada tiap *window* untuk dijadikan *fingerprint* tiap dokumen (Ridho, 2013). Dari contoh window pada tahap 4 diatas, maka *fingerprint* yang diperoleh
   `[2177170647,1],[2254381403,4],[2204434801,8],[2114375646,12],[2079779016,16],[2084757836,17],[2093449587,21],[2235134110,22],[2293151702,23],[2432429448,27],[2105452718,28],[2187519307,31],[2100208314,33],[2095121226,34],[2155273004,37]`

2.5.1 Konsep Biword pada Winnowing

Biword merupakan suatu teknik *matching* pada proses tokenisasi atau pemisahan kata (*fingerprint*) pada teks dokumen. Konsep *biword* itu sendiri menyerap teknik *phrase-based.* Pada tahap awal akan dilakukan konversi setiap dokumen untuk satu set *bigram* (dua kata) (Kok dan Salim, 2010). Secara umum konsep *biword* pada *winnowing* tidaklah berbeda jauh dengan konsep *winnowing* yang aslinya, perbedaannya hanya terletak pada penggunaan *biword* yang menggantikan metode *k-gram* dan adanya penggunaan fungsi *MD5* untuk proses enkripsi frasa sebelum dilakukan *rolling hash*. Adanya proses enkripsi frasa dengan fungsi *MD5* agar setiap frasa yang memiliki panjang karakter yang berbeda-beda menjadi karakter yang memiliki panjang yang sama dengan panjang 32 karakter. *MD5* dipilih untuk enkripsi frasa karena memiliki panjang karakter yang mendekati panjang karakter yang terbaik dibandingkan panjang karakter enkripsi *SHA* (40 karakter) (Ridho, 2013). Berikut ini adalah langkah-langkah penerapan *biword* pada *winnowing* (Ridho, 2013).

1. Melakukan pembersihan teks.
   Misalnya kalimat “Teknik Informatika adalah jurusan terbaik di UIN SUSKA” dilakukan pembersihan teks maka menjadi “teknik informatika adalah jurusan terbaik di uin suska”
2. Melakukan pemotongan teks menjadi *biword* yang kemudian dienkripsi menggunakan *MD5*.

Biword yang diperoleh dari kalimat sebelumnya adalah

```
[0] => teknik informatika
[1] => informatika adalah
[2] => adalah jurusan
[3] => jurusan terbaik
[4] => terbaik di
[5] => di uin
[6] => uin suska
```

Kemudian dienkripsi dengan *MD5* maka diperoleh

```
[0] => 5495ef670ce7ce89fae9677affba8b26
[1] => 7994e4e6ae098e4c6b3253c807369af9
[2] => bc4680f26e956ebd658eff70b4c43963
[3] => 6f8e3f87b7d6f0fd6e2a713317f11e52
[4] => f8f8b58118825324b99b48b94bddaf6d
[5] => 80c767e6cf7ff443a3d2550e1f8b2015
[6] => c0759ebcb99f6e643883261b366d3f12
```

3. Menghitung nilai *hash* menggunakan *rolling hash* (Persamaan 2.7). Misal dalam *rolling hash* menggunakan b=2 dan k=32 maka diperoleh nilai hash :

```
[0] => 238798777778
[1] => 246686918097
[2] => 375295264443
[3] => 300564078280
[4] => 369249503152
[5] => 256509698823
[6] => 328629556164
```

4. Membentuk *window* dengan ukuran *w*. Misal dibentuk dengan ukuran *window* w=4 diperoleh

```
238798777778,246686918097,375295264443,300564078280
246686918097,375295264443,300564078280,369249503152
375295264443,300564078280,369249503152,256509698823
300564078280,369249503152,256509698823,328629556164
```

5. Pemilihan beberapa nilai *hash* menjadi *document fingerprinting*. Maka diperoleh :

```
[238798777778,0], [246686918097,1], [256509698823,5]
```

2.5.2 Jaccard Coefficient

Jaccard Coefficient merupakan persamaan yang digunakan untuk menentukan tingkat kemiripan antara dua dokumen teks pada algoritma *winnowing*. Langkah ini, dilakukan setelah melakukan perhitungan nilai *hash* dan

memilih *fingerprint* yang terkecil dari dua dokumen teks (Ridho, 2013). Berikut persamaan *jaccard coefficient*:

$$\text{Similaritas}(d_i,d_j) = \frac{|W(di) \cap W(dj)|}{|W(di) \cup W(dj)|} \times 100\% \quad \text{……..(2.9)}$$

Keterangan:

$W(d_i)$: *fingerprint* terkecil dokumen teks 1

$W(d_j)$: *fingerprint* terkecil dokumen teks 2