BAB II

LANDASAN TEORI

2.1 Plagiarisme

Kurniawati dan Wicaksana (2008) menjelaskan bahwa: “Plagiarisme adalah tindakan penyalahgunaan, pencurian/perampasan, penerbitan, pernyataan, atau menyatakan, sebagai milik sendiri sebuah pikiran, ide, tulisan, atau ciptaan yang sebenarnya milik orang lain.”. Selain itu, plagiarisme juga dijelaskan sebagai tindakan-tindakan pencurian hasil karya orang lain dan menjadikan hasil karya tersebut seolah-olah hasil karyanya sendiri (Setiyati, 2015).

Menurut Parvati dan Abhipsita dalam Lede, dkk. (2014), plagiarisme dikelompokkan ke dalam enam bentuk, yaitu:

1. Plagiarisme kata per kata; penyalinan kalimat secara langsung dari sebuah dokumen teks tanpa adanya pengutipan atau perizinan.
2. Plagiarisme *authorship*; pengakuan hasil karya orang lain sebagai milik sendiri dengan mencantumkan nama sendiri menggantikan nama pengarang yang sebenarnya.
3. Plagiarisme ide; penggunaan ulang suatu gagasan atau pemikiran asli dari sebuah sumber teks tanpa bergantung bentuk teks sumber.
4. Plagiarisme sumber sekunder; pengutipan sumber asli yang didapat dari sumber sekunder dengan menghiraukan teks asli dari sumber yang sebenarnya.
5. Plagiarisme struktur sumber; penyalinan atau penjiplakan struktur suatu argumen dari sebuah sumber.

6. Plagiarisme *paraphrase*; Penulisan ulang dengan mengubah kata atau sintaksis, tetapi teks aslinya masih dapat dikenali.

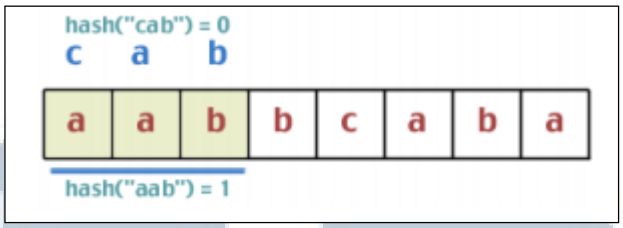
Berdasarkan proporsi dan tingkat kemiripan dokumen, plagiarisme diklasifikasikan sebagai berikut (Ariyani, dkk. 2016).

1. Plagiarisme ringan, apabila tingkat kemiripan di bawah 30 persen (<30%).
2. Plagiarisme sedang, apabila tingkat kemiripan antara 30 sampai dengan 70 persen.
3. Plagiarisme berat, apabila tingkat kemiripan di atas 70 persen (>70%).

2.2 Algoritma Rabin-Karp

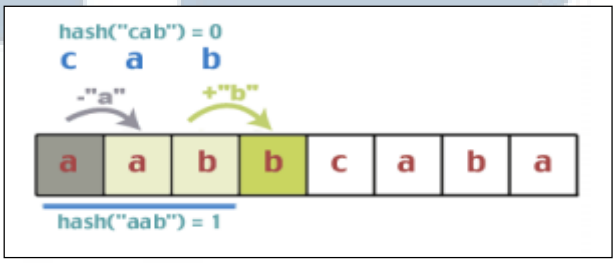
Algoritma Rabin-Karp adalah algoritma pencocokkan *string* yang ditemukan oleh Michael Rabin dan Richard Karp. Algoritma ini membandingkan nilai *hash* antara *string* masukan dengan *substring* pada teks, apabila *hash value* keduanya sama, maka akan dilakukan perbandingan sekali lagi terhadap karakter-karakternya, apabila *hash value* keduanya tidak sama, maka *substring* akan bergeser ke kanan. Perhitungan nilai *hash* yang efisien saat dilakukan pergeseran akan memengaruhi performa algoritma ini (Firdaus, 2003).

Berikut ini dijelaskan ilustrasi mengenai cara kerja algoritma Rabin-Karp yang diambil dari Firdaus (2003). Diberikan masukan "cab" dan teks "aabbcaba". Fungsi *hash* yang dipakai misalnya akan menambahkan nilai keterurutan setiap huruf dalam *alphabet* (a = 1, b = 2, dan seterusnya) dan melakukan *modulo* dengan 3. Didapatkan nilai *hash* "cab" adalah 0 dan tiga karakter pertama pada teks yaitu "aab" adalah 1.



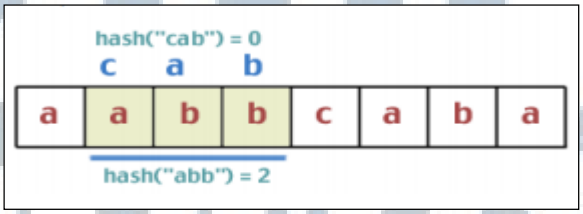
Gambar 2.1 *Fingerprint* Awal
(Firdaus, 2003)

Hasil perbandingan ternyata tidak sama, maka *substring* pada teks akan bergeser satu karakter ke kanan. Algoritma tidak menghitung kembali nilai *hash substring*. Di sinilah 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 (Firdaus, 2003).



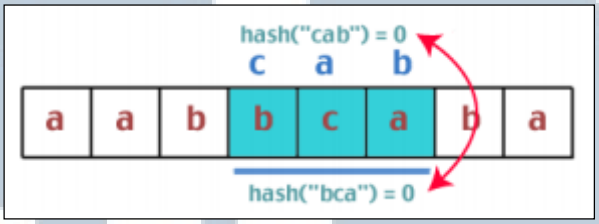
Gambar 2.2 Menggeser *Fingerprint*
(Firdaus, 2003)

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



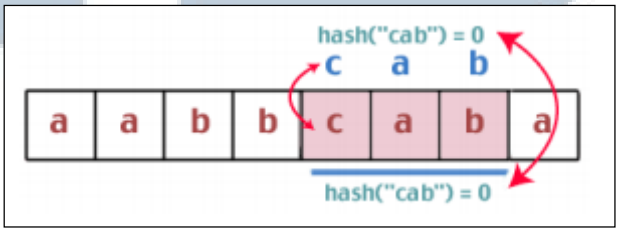
Gambar 2.3 Perbandingan Kedua
(Firdaus, 2003)

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



Gambar 2.4 Perbandingan Keempat (Nilai *Hash* Sama)
(Firdaus, 2003)

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 2.5 Perbandingan Kelima (*String* Ditemukan)
(Firdaus, 2003)

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 iterasi yang dilakukan untuk menemukan solusi. Hasil ini jauh lebih baik daripada kompleksitas waktu yang didapat menggunakan algoritma *brute-force* yaitu O(mn) (Firdaus, 2003).

2.3 *Hash*

Hashing merupakan sebuah cara untuk mengubah sebuah *string* menjadi suatu nilai *hash* unik dengan panjang tertentu (Firdaus, 2003). Fungsi *hash* yang sederhana masih menghasilkan permasalahan seperti *string* berbeda namun menghasilkan nilai *hash* sama. salah satu fungsi *hash* yang dapat mencegah hal tersebut adalah *Rolling Hash*. *Rolling Hash* adalah fungsi *hash* dengan basis dan nilai pangkat tertentu. Nilai basis yang digunakan biasanya adalah bilangan prima yang cukup besar. Persamaannya adalah sebagai berikut.

$$H = c_1 * a^{k-1} + c_2 * a^{k-2} + c_3 * a^{k-3} \ldots + c_k * a^0 \quad \ldots(2.1)$$

dengan H merupakan nilai *hash*, c nilai ASCII suatu karakter, a nilai basis, dan k jumlah karakter. Sebagai contoh akan didapatkan nilai *hash string* "abc" sebesar 12914 dengan nilai basis 11 melalui perhitungan 97 (nilai ASCII a) x 11^2 + 98 (nilai ASCII b) x 11^1 + 99 (nilai ASCII c) x 11^0 (Wirawan, 2003).

2.4 N-Gram

N-Gram merupakan sebuah cara untuk mendapatkan potongan karakter sebanyak n dari sebuah kalimat berdasarkan jumlah n yang ditentukan (Cavnar & Trenkle, 1994). Contoh dari penggunaan N-Gram dapat dilihat pada Tabel 2.1 dengan contoh kalimat "buku saya", dimana "_" direpresentasikan sebagai spasi.

Tabel 2.1 Contoh Pemotongan N-Gram Pada Kalimat

Gram	Hasil Pemotongan
Uni-Gram	b, u, k, u, _, s, a, y, a
Bi-Gram	bu, uk, ku, u_, _s, sa, ay, ya
Tri-Gram	buk, uku, ku_, u_s, _sa, say, aya
Quad-Gram	buku, uku_, ku_s, u_sa, _say, saya

2.5 *Dice's Similarity Coefficient*

Perhitungan tingkat *similarity* pada penelitian ini menggunakan *Dice's Similarity Coefficient.*

$$S = \frac{2C}{A + B} \quad \ldots(2.2)$$

dimana S merupakan nilai *similarity*, A dan B adalah jumlah kata yang terdapat pada teks pertama dan kedua, dan C adalah jumlah kata sama yang didapat dari hasil perbandingan masing-masing kata yang terdapat dari dokumen pertama dan dokumen kedua (Kosinov, 2001).

2.6 Algoritma *Confix-Stripping Stemmer*

Berdasarkan Adriani, dkk. dalam Kurniawan, dkk. (2012) algoritma *Confix-Stripping stemmer* adalah algoritma yang digunakan untuk melakukan proses *stemming* terhadap kata-kata berimbuhan. Algoritma *Confix-Stripping stemmer* memiliki aturan imbuhan dengan model sebagai berikut.

[[[AW +]AW +]AW +] Kata-Dasar [[+AK][+KK][+P]

AW : Awalan AK : Akhiran

KK : Kata ganti kepunyaan P : Partikel

Langkah-langkah algoritma *Confix-Stripping stemmer* adalah sebagai berikut (Kurniawan, dkk., 2012).

1. Kata yang belum di-*stemming* dibandingkan ke dalam *database* kamus kata dasar. Jika ditemukan, maka kata tersebut diasumsikan sebagai kata dasar dan algoritma berhenti. Jika kata tidak ada maka lanjut ke langkah 2.
2. Jika kata di-*input* memiliki pasangan awalan-akhiran "be-lah","be-an","me-i","di-i","pe-i", atau "te-i" maka langkah *stemming* selanjutnya adalah 5, 3, 4,

5, 6, tetapi jika kata yang di-*input* tidak memiliki pasangan awalan-akhiran tersebut, langkah *stemming* berjalan normal yaitu 3, 4, 5, 6.

3. Hilangkan partikel dan kata ganti kepunyaan. Pertama, hilangkan partikel (“-lah”, “-kah”, “-tah”, “-pun”). Setelah itu hilangkan juga kata ganti kepunyaan (“-ku”, “-mu”, atau “-nya”). Contoh: kata “bajumulah”, proses *stemming* pertama menjadi “bajumu” dan proses *stemming* kedua menjadi “baju”. Jika kata “baju” ada di dalam kamus maka algoritma berhenti. Sesuai dengan model imbuhan, menjadi:

   [[[AW+]AW+]AW+] Kata Dasar [+AK]

4. Hilangkan juga akhiran (“-i”,”-an”, dan “-kan”), sesuai dengan model imbuhan, maka menjadi:

   [[[AW+]AW+]AW+] Kata Dasar

5. Penghilangan awalan (“be-“, ”di-“, ”ke-“, ”me-”, ”pe-”, ”se-”, dan “te-“) mengikuti langkah-langkah berikut.
   a. Algoritma akan berhenti jika:
      i. Awalan diidentifikasi bentuk sepasang imbuhan yang tidak diperbolehkan dengan akhiran (berdasarkan Tabel 2.2) yang dihapus pada langkah 3.
      ii. Diidentifikasi awalan yang sekarang identik dengan awalan yang telah dihapus sebelumnya.
      iii. Kata tersebut sudah tidak memiliki awalan.
   b. Identifikasi jenis awalan dan peluruhannya bila diperlukan. Jenis awalan ditentukan dengan aturan:

i. Jika awalan dari kata adalah “di-“, “ke-“, atau “se“ maka awalan dapat langsung dihilangkan.

ii. Hapus awalan “te-“, “be-“, “me-“, atau “pe-“ yang menggunakan aturan peluruhan yang dijelaskan pada Tabel 2.3. Sebagai contoh kata “menangkap”, setelah menghilangkan awalan “me-“ maka kata yang didapat adalah “nangkap”. Karena kata “nangkap” tidak ditemukan dalam *database* kata dasar maka karakter “n” diganti dengan karakter “t” sehingga dihasilkan kata “tangkap” dan kata “tangkap” merupakan kata yang sesuai dengan kata yang ada di *database* kata dasar, maka algoritma berhenti.

6. Jika semua langkah gagal, maka kata yang diuji pada algoritma ini dianggap sebagai kata dasar.

Tabel 2.2 Kombinasi Prefiks dan Sufiks yang Tidak Diperbolehkan
(Adriani, dkk. dalam Kurniawan, dkk., 2012)

Awalan (Prefiks)	**Akhiran (Sufiks)**
be-	-i
di-	-an
ke-	-i, -kan
me-	-an
se-	-i, -kan
te-	-an

Tabel 2.3 Aturan Peluruhan Kata Dasar
(Tahitoe dan Purwitasari, 2010)

Aturan	**Bentuk Awalan**	**Peluruhan**
1	berV...	ber-V… | be-rV…
2	berCAP…	berCAP… dimana C!=‟r‟ & P!=‟er‟
3	berCAerV...	ber-CaerV... dimana C!=‟r‟
4	belajar...	bel-ajar
5	beC_1erC_2…	be-C_1erC_2…dimana C_1!={'r'|'l'}
6	terV…	ter-V…|te-rV…
7	terCerV…	ter-CerV…dimana C!='r'
8	terCP	ter-CP... dimana C!=‟r‟ dan P!=‟er‟
9	teC_1erC_2	te-C_1erC_2…dimana C_1!='r'

Tabel 2.3 Aturan Peluruhan Kata Dasar (lanjutan)
(Tahitoe dan Purwitasari, 2010)

Aturan	Bentuk Awalan	Peluruhan
10	me{l|r|w|y}V…	me-{l|r|w|y}V…
11	mem{b|f|v}…	mem-{b|f|v}…
12	mempe{r|l}...	mem-pe…
13	mem{rV|V}…	me-m{rV|V}…|me-p{rV|V}…
14	men{c|d|j|z}…	men-{c|d|j|z}…
15	menV…	me-nV…|me-tV…
16	meng{g|h|q}…	meng-{g|h|q}…
17	mengV…	meng-V…|meng-kV…
18	menyV…	meny-sV…
19	mempV…	mem-pV… dimana V!='e'
20	pe{w|y}V…	pe-{w|y}V…
21	perV…	per-V…|pe-rV…
22	perCAP	per-CAP... dimana C!="r" dan P!="er"
23	perCAerV	per-CAerV... dimana C!="r"
24	pem{b|f|V}…	pem-{b|f|V}…
25	pem{rV|V}…	pe-m{rV|V}…|pe-p{rV|V}…
26	pen{c|d|j|z}…	pen-{c|d|j|z}…
27	penV…	pe-nV…|pe-tV…
28	peng{g|h|q}	peng-{g|h|q}…
29	pengV	peng-V…|peng-kV…
30	penyV…	peny-sV…
31	pelV..	pe-lV…; kecuali untuk kata "pelajar" menjadi "ajar"
32	peCP	pe-CP…dimana C!={r|w|y|l|m|n} dan P!='er'
33	perCerV	Per-CerV… dimana C!={r|w|y|l|m|n}

Keterangan:
C : Huruf konsonan
V : Huruf vokal
A : huruf vokal atau konsonan
P : partikel atau fragmen dari suatu kata, misalnya "er"

Tabel 2.4 Modifikasi dan Tambahan Aturan pada Tabel 2.3
(Tahitoe dan Purwitasari, 2010)

Aturan	Format Kata	Pemenggalan
12	mempe…	mem-pe...
16	meng{g|h|q|k}...	meng-{g|h|q|k}...
34	terC1erC2...	ter-C1erC2... dimana C1!= ,r"
35	peC1erC2...	pe-C1erC2... dimana C1!={r|w|y|l|m|n}

2.7 *Confusion Matrix*

Dalam pengukuran akurasi performa sistem diperlukan suatu rumus untuk melakukan pengukuran performa sistem tersebut. Saat pengukuran dilakukan berdasarkan klasifikasi tertentu, maka pengukuran performa dapat menggunakan *Confusion Matrix* . Dalam *Confusion Matrix* terdapat beberapa jenis nilai klasifikasi entri (Maimon dan Rokach dalam Riany, dkk., 2016):

Tabel 2.5 *Confusion Matrix*

		Predicted	
		Positive	*Negative*
Actual	*Positive*	TP	FN
	Negative	FP	TN

a. *True positive* (TP) : jumlah *record* positif yang diklasifikasikan sebagai positif.

b. *False positive* (FP) : jumlah *record* negatif yang diklasifikasikan sebagai positif.

c. *False negative* (FN): jumlah *record* positif yang diklasifikasikan sebagai negatif.

d. *True negative* (TN): jumlah *record* negatif yang diklasifikasikan sebagai negatif.

Berdasarkan klasifikasi entri diatas, maka selanjutkan dapat dilakukan perhitungan seperti *Precision, Recall, F Measure,* dan *Accuracy.*

$$Precision = \frac{TP}{(TP + FP)} \quad \ldots(2.3)$$

Rumus 2.3 merupakan persamaan *Precision*. *Precision* berfungsi untuk mengukur tingkat ketepatan antara informasi yang diminta oleh *user* dengan jawaban yang diberikan sistem.

$$Recall = \frac{TP}{(TP + FN)} \quad \ldots(2.4)$$

Rumus 2.4 merupakan persamaan *Recall*. *Recall* berfungsi untuk mengukur tingkat keberhasilan sistem dalam menemukan kembali sebuah informasi.

$$F\ Measure = 2x \frac{Recall\ x\ Precision}{Recall + Precision} \quad \ldots(2.5)$$

Rumus 2.5 merupakan persamaan *F Measure*. *F Measure* merupakan sebuah rumus tunggal yang didapat dengan menggabungkan rumus *Precision* dan *Recall* (Destuardi dan Sumpeno dalam Riany, dkk., 2016).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad \ldots(2.6)$$

Rumus 2.6 merupakan persamaan *Accuracy*. Accuracy adalah jumlah data yang diklasifikasikan benar dibagi jumlah seluruh data (Pratomo, 2013).