BAB II

TINJAUAN PUSTAKA

2.1 Plagiarisme

Plagiarisme merupakan praktik penyalahgunaan hak kekayaan intelektual milik orang lain dan karya tersebut diakui secara tidak sah sebagai hasil karya pribadi (Sulianta, 2007). Di samping itu, menurut Kamus Besar Bahasa Indonesia, plagiarisme adalah penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri.

Pelaku tindakan plagiarisme disebut dengan plagiator. Plagiator dapat dijatuhi hukuman pidana karena terdapat Undang-Undang yang membahas tentang tindakan plagiarisme. Dengan adanya Undang-Undang tersebut, dalam dunia pendidikan, baik di sekolah maupun perguruan tinggi, hukuman pun akan diberlakukan bagi para plagiator dalam dunia akademik (Yahman, 2012).

Dari segi kesengajaan, plagiarisme dibagi menjadi dua jenis, yaitu plagiarisme secara tidak sengaja dan plagiarisme secara sengaja. Plagiarisme secara tidak sengaja terjadi ketika seseorang menggunakan kata-kata atau gagasan milik orang lain tetapi lupa atau tidak mengetahui bagaimana caranya untuk memberi tahu narasumbernya. Contohnya adalah mengambil kalimat buatan orang lain tetapi lupa menambahkan tanda petik atau lupa menyebutkan dari mana kalimat tersebut diambil. Berbeda dengan plagiarisme secara sengaja, yang mana seseorang mengambil kata-kata atau gagasan dari orang lain dan mengakui

sebagai miliknya pribadi. Hukuman untuk tindakan plagiarisme secara tidak sengaja lebih ringan dibandingkan plagiarisme secara sengaja. Seseorang yang merupakan siswa atau mahasiswa dapat dikeluarkan dari sekolah atau kampus karena tindakan plagiarisme ini (Kirszner & Mandell, 2008).

Untuk dapat mendeteksi plagiarisme, terdapat tiga metode yaitu perbandingan teks lengkap, dokumen *fingerprinting*, dan kesamaan kata kunci. Berikut ini adalah penjelasan dari masing-masingmetode pendeteksi plagiarisme (Kurniawati & Wicaksana, 2008).

1. Perbandingan Teks Lengkap

   Metode ini diterapkan dengan membandingkan semua isi dokumen uji dengan dokumen asli dan dapat diterapkan pada dokumen berukuran besar. Walaupun metode ini membutuhkan waktu yang lama, hasilnya cukup efektif.

2. Dokumen *Fingerprinting*

   Dokumen *fingerprinting* merupakan metode yang digunakan untuk mengetahui persentase kemiripan antar dokumen, baik semua teks yang terdapat di dalam dokumen atau hanya bagian dari teks (*substring*). Prinsip kerja dari metode ini adalah dengan menggunakan teknik *hashing*.

3. Kesamaan Kata Kunci

   Prinsip dari metode ini adalah mengekstrak kata kunci dari dokumen. Kata-kata yang tidak penting akan dihapus. Kemudian, dibandingkan dengan kata kunci pada dokumen yang lain.

Dalam mendeteksi plagiarisme, algoritma yang digunakan harus memenuhi tiga kriteria. Berikut adalah ketiga kriteria tersebut (Schleimer, Wilkerson, & Aiken, 2003).

1. *Whitespace Insensitivity*

   Dalam mencocokkan *file* teks, perbandingan tidak boleh terpengaruh oleh hal-hal seperti ekstra spasi, huruf besar, tanda baca, dan lain-lain.

2. *Noise Suppression*

   Penemuan kata-kata yang pendek seperti kata ‘the’ dalam bahasa Inggris dan ‘yang’ dalam bahasa Indonesia tidak memiliki makna. Setiap penemuan kata harus cukup besar untuk memberitahukan bahwa kata ini telah disalin dan bukan kata umum.

3. *Position Independence*

   Urutan kata dalam dokumen seharusnya tidak memengaruhi kecocokan yang ditemukan. Selain itu, menghapus bagian dari dokumen seharusnya tidak memengaruhi kecocokan.

2.2 Dokumen Elektronik

Dokumen elektronik adalah informasi yang dibuat, diteruskan, dikirimkan, diterima dan disimpan dalam bentuk analog, digital atau sejenisnya, yang dapat dilihat, ditampilkan, atau didengar melalui komputer atau alat elektronik lainnya seperti *smartphone* dan tablet PC. Isi dari dokumen tidak terbatas pada tulisan, suara, gambar atau sejenisnya. Dengan menggabungkan beberapa elemen tersebut, informasi yang ingin disampaikan akan lebih jelas (Riadi, 2008).

Dari sekian banyak jenis dokumen elektronik yang dikenal, yang umum dipakai adalah *text file* (.txt), Microsoft Word Document(.doc), Microsoft Word Open XML Document (.docx), dan *Portable Document Format*(.pdf). *Text file* merupakan sebuah *file* yang hanya mengandung karakter standar dan tidak memiliki kode pemformatan atau *tags.* Selain itu, dalam sebuah *text file* tidak bisa diberikan gambar dan tidak mendukung aksi *undo* (American Heritage Dictionaries, 2006).

Berbeda dengan *text file,* .doc dan .docx mampu menyimpan segala informasi terkait dengan teks, gambar, *layout,* pemformatan dan sebagainya. Kedua dokumen ini bisa dibuka dan diolah menggunakan Microsoft Word. Kelebihan lainnya adalah bisa melakukan aksi *undo.* Yang membedakan *file* .doc dengan .docx adalah versi Microsoft Word yang membuatnya dan ukuran *file* yang dihasilkan (Gookin, 2013).

Portable Document Format atau PDF merupakan format *file* yang menangkap seluruh elemen dari sebuah dokumen dan merepresentasikannya sebagai dokumen elektronik yang bisa dilihat, dinavigasi, dicetak, dan diteruskan ke orang lain. Untuk membuatnya, diperlukan sebuah *software* yang merupakan salah satu produk dari Adobe Acrobat. PDF yang sangat berguna untuk dokumen seperti artikel majalah, brosur, atau selebaran karena mampu mempertahankan penampilan asli. Selain itu, PDF dapat men-*embed font* sehingga dapat dilihat di media mana saja (Rouse, 2010).

2.3 Bahasa Indonesia

Bahasa Indonesia merupakan bahasa resmi Republik Indonesia dan bahasa persatuan bangsa Indonesia. Bahasa Indonesia diresmikan penggunaannya setelah proklamasi kemerdekaan Indonesia. Seiring berjalannya waktu, bahasa Indonesia mengalami beberapa penyempurnaan ejaan. Saat ini, bahasa Indonesia sudah mengikuti aturan ejaan bahasa Indonesia yang disempurnakan atau EYD.

EYD adalah ejaan bahasa Indonesia yang berlaku sejak 1972. Ejaan ini menggantikan ejaan sebelumnya, yaitu Ejaan Republik. Keberadaan EYD merupakan salah satu upaya untuk menstandarkan penulisan bahasa Indonesia secara baik dan benar. Berikut adalah perbedaan EYD dengan ejaan sebelumnya (Waridah, 2008).

1. 'tj' menjadi 'c' : tjutji → cuci
2. 'dj' menjadi 'j' : djarak → jarak
3. 'j' menjadi 'y' : sajang → sayang
4. 'nj' menjadi 'ny' : njamuk → nyamuk
5. 'sj' menjadi 'sy' : sjarat → syarat
6. 'ch' menjadi 'kh' : achir → akhir
7. awalan 'di-' dan kata depan 'di' dibedakan penulisannya. Kata depan 'di' pada contoh "di rumah", "di sawah", penulisannya dipisahkan dengan spasi. Untuk 'di-' pada “dibeli” dan“dimakan” ditulis serangkai dengan kata yang mengikutinya.

2.4 Pencocokan String

Pencocokan *string* atau *string matching* adalah suatu proses mencari satu atau beberapa pola teks dengan mencocokkan pola teks dengan masing-masing bagian kecil dari teks yang lebih besar. Banyaknya penggunaan elemen multimedia teks untuk menyampaikan informasi menjadikan pencocokan *string* menjadi bahasan yang penting dalam dunia teknologi informasi (Lecroq & Charras, 2004).

Pencocokan *string* dapat dimanfaatkan dalam banyak hal, misalkan pencarian suatu kata dalam dokumen, penerjemah kata dengan mencocokkan kata dari kamus, dan pendeteksi plagiarisme dengan menghitung kecocokan teks. Persoalan pencarian *string* dirumuskan sebagai berikut (Munir, 2004).

1. Teks (*text*), yaitu (*long*) *string* yang panjangnya *n* karakter.
2. Pola (*pattern*), yaitu *string* dengan panjang *m* karakter ($m < n$) yang akan dicari di dalam teks.

Di bawah ini adalah contoh dari pencocokan *string* (Munir, 2004).

Pola : hari

Teks : kami pulang hari ini

⇑ target

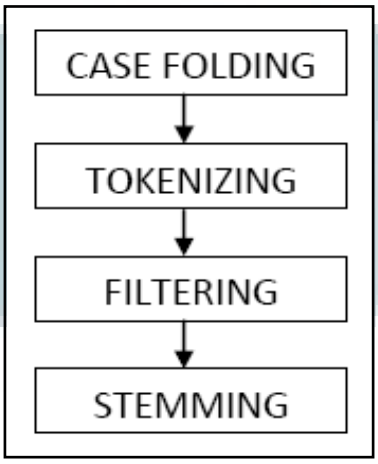
Dalam pencocokan *string*, penggunaan algoritma yang tepat dapat memberikan hasil yang tepat, akurat, dan sesuai keinginan pengguna. Banyak jenis algoritma pencocokan *string* yang ditawarkan. Beberapa contoh yaitu algoritma *brute force*, algoritma Boyer-Moore, algoritma Knuth-Morris-Pratt, dan algoritma Rabin-Karp. Penggunaan algoritma pencocokan *string*

bergantung pada jenis pencarian. Misalkan untuk pencocokan banyak pola teks, sebaiknya menggunakan algoritma Rabin-Karp dibandingkan algoritma lainnya (Cormen, Leiserson, Rivest, & Stein, 2003).

2.5 Text Mining

Text mining merupakan sekumpulan dari teknik atau metode yang digunakan untuk memproses data berupa teks dalam ukuran yang besar. Tujuannya adalah untuk mengekstrak dan menata konten dari teks tersebut. Hasil ekstrak dan penataan tersebut akan digunakan untuk melakukan analisis pesat dan menemukan data baru yang sebelumnya tersembunyi (Tufféry, 2011).

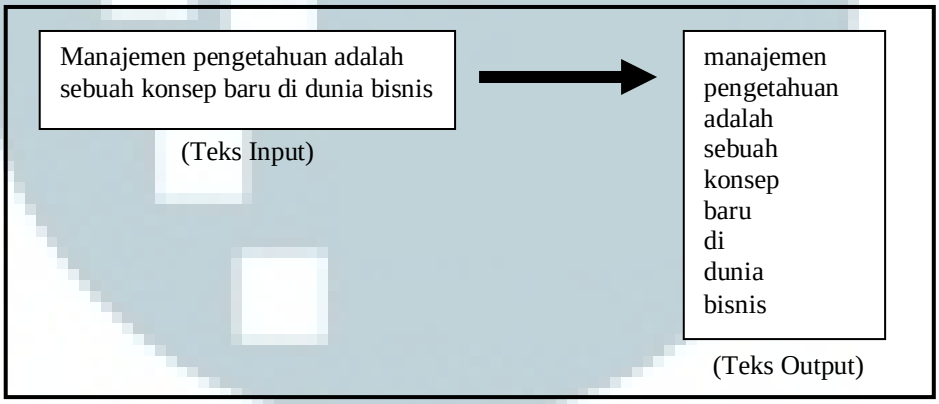
Pada *text mining*, informasi yang akan digali berasal dari data yang memiliki struktur yangtidak beraturan. Oleh karena itu, diperlukan tahap-tahap untuk mengubah bentuk menjadi data yang terstruktur sesuai kebutuhannya sehingga dapat diolah lebih lanjut. Tahap-tahap ini sering disebut *pre-processing*. *Pre-processing* terdiri dari *case folding, tokenizing, filtering,* dan *stemming* (Triawati, 2009).



Gambar 2.1 Tahap *Pre-processing*(Triawati, 2009)

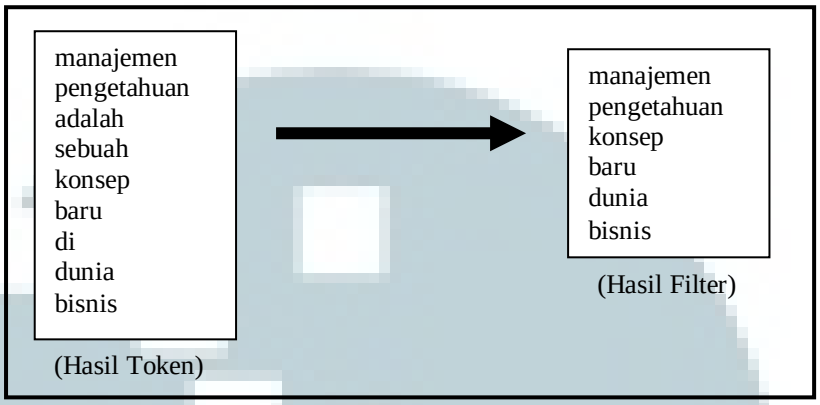
Tahap pertama dalam *text pre-processing* adalah *case folding*. Tujuannya adalah mengubah semua huruf dalam dokumen menjadi huruf kecil. Hanya huruf ‘a’ sampai dengan ‘z’ yang diterima. Karakter selain huruf dihilangkan dan dianggap *delimiter*.

Setelah tahap *case folding* selesai, proses dilanjutkan dengan *tokenizing*. Dalam tahap ini terjadi pemotongan *string* input berdasarkan tiap kata yang menyusunnya sehingga dihasilkan *array* yang berisi kata-kata. Contoh dari tahap ini digambarkan sebagai berikut (Triawati, 2009).



Gambar 2.2 Tahap *Case Folding* dan *Tokenizing* (Triawati, 2009)

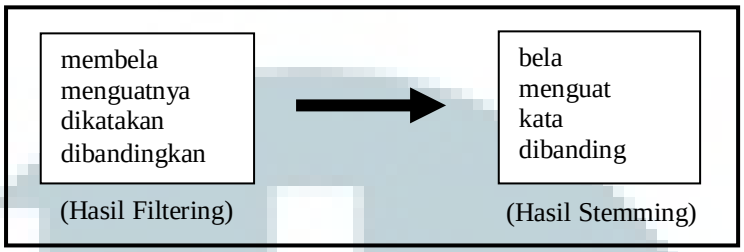
Tahap *filtering* adalah tahap mengambil kata-kata penting dari hasil tahap sebelumnya. Ada berbagai cara bisa digunakan untuk *filtering*, yaitu dengan algoritma *stop list* (membuang kata yang kurang penting) dan *word list* (menyimpan kata penting). *Stop list* atau *stop word* adalah kata-kata yang tidak deskriptif yang dapat dibuang, seperti “yang”, “dan”, “di”, dan “dari”. Berikut adalah contoh dari tahap *filtering* (Triawati, 2009).



Gambar 2.3 Tahap *Filtering* (Triawati, 2009)

Kata-kata hasil *filtering* akan memasuki tahap *stemming*. *Stemming* merupakan suatu proses yang terdapat dalam sistem *information retrieval* yang mengubah kata-kata dalam suatu dokumen ke kata-kata akarnya (*root word*) dengan menggunakan aturan-aturan tertentu sehingga sesuai dengan struktur morfologi bahasa Indonesia yang baik dan benar. Proses *stemming* pada teks berbahasa Indonesia berbeda dengan *stemming* pada teks berbahasa Inggris. Pada teks berbahasa Inggris, proses yang diperlukan hanya proses menghilangkan *suffix*, sedangkan pada teks berbahasa Indonesia yang harus dihilangkan adalah *suffix*, *prefix*, dan konfiks (Agusta, 2009).

Hasil dari proses *stemming* bergantung pada bahasa yang diaplikasikan. Semakin kompleks struktur morfologi yang miliki suatu bahasa, semakin akurat hasil *stemming* yang akan diperoleh. Bahasa Indonesia memiliki keuntungan untuk dilakukan proses *stemming* karena struktur morfologinya yang kompleks. Berikut adalah contoh dari *stemming* pada teks berbahasa Indonesia (Asian, 2007).



Gambar 2.4 Tahap *Stemming* (Triawati, 2009)

2.6 Algoritma Rabin-Karp

2.6.1 Definisi Algoritma Rabin-Karp

Algoritma Rabin-Karp merupakan salah satu dari banyak algoritma *string matching* yang diciptakan oleh Michael O. Rabin dan Richard M. Karp. Prinsip kerja dari algoritma ini adalah mencari pola (*pattern*) teks dengan membandingkan nilai *hash* dari setiap teks. Untuk teks dengan panjang *n* dan pola pencarian dengan panjang *m*, kompleksitas waktu rata-rata dan paling baik adalah O(n), sedangkan yang paling buruk adalah O(mn). Dengan kompleksitas waktu yang dihasilkan, algoritma ini tidak baik digunakan untuk mencari satu pola saja. Oleh karena itu, algoritma Rabin-Karp lebih baik digunakan untuk mencari banyak pola. Algoritma ini memiliki kelebihan yaitu bisa mencari salah satu dari pola sebanyak *k* dengan kompleksitas waktu rata-rata O(n), tanpa memerhatikan nilai *k*. Implementasi paling mudah dari algoritma Rabin-Karp adalah untuk mendeteksi plagiarisme pada teks. Dengan adanya kemampuan untuk mencari banyak pola, persentase kemiripan teks dapat dihitung dengan mudah (Gupta, Agarwal, & Varshney, 2010).

2.6.2 Konsep Algoritma Rabin-Karp

Berikut adalah karakteristik utama dari algoritma Rabin-Karp (Gupta, Agarwal, & Varshney, 2010).

1. Menggunakan fungsi *hashing*.
2. Fase *pre-processing* menggunakan kompleksitas waktu O(m).
3. Waktu yang diperlukan adalah O(n + m).
4. Untuk fase pencarian, kompleksitasnya adalah O(mn).

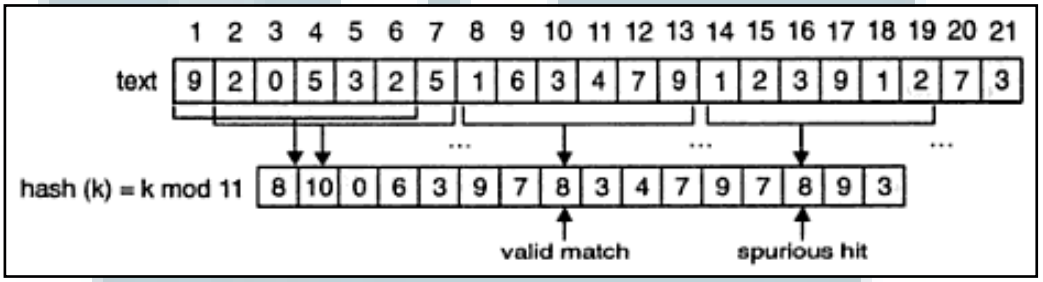
Secara garis besar, algoritma Rabin-Karp dapat dijelaskan dengan *pseudocode* berikut.

```
RabinKarpMatcher(string P, string T, integer d,
integer q)
   n ← length[T]; m ← length[P]
   h ← d^(m-1) mod q
   p = 0; t_n = 0
   for i ← 1 to m do
      p ← (d p + P[i]) mod q
      t_n ← (d t_n + T[i]) mod q
   for s ← 0 to n – 1 do
      if p = t_s then
         if P[1..m]= T[s+1..s+m] then
            print "Pattern found."
      if s < n – 1 then
         t_s ← (d(t_s – T[s+1]h) + T[s+m+1]) mod q
```

Gambar 2.5 *Pseudocode* Algoritma Rabin-Karp (Gupta, 2010)

Konsep pencarian dan pencocokan *string* dengan algoritma Rabin-Karp adalah membandingkan nilai *hash* pola dengan nilai *hash substring* teks dengan panjang sama dengan pola. Salah satu alasan mengapa algoritma Rabin-Karp menggunakan teknik *hashing* adalah untuk mengurangi waktu pencarian. Lebih cepat mengubah suatu *substring* sepanjang polayang dicari menjadi nilai *hash* dan

kemudian dibandingkan, dari pada membandingkan per karakter dari teks. Ketika sudah didapatkan nilai *hash*, algoritma ini secara berulang akan mencari seluruh *substring* dengan panjang yang sama dengan pola yang dicari dan mengubahnya menjadi nilai *hash*. Perbandingan per karakter akan dilakukan jika dan hanya jika nilai *hash* antara *substring* dan pola yang dicari adalah sama. Berikut adalah ilustrasi dari pencarian dan pencocokan *string* oleh algoritma Rabin-Karp (Wang, 2008).



Gambar 2.6 Ilustrasi Algoritma Rabin-Karp (Gupta, 2010)

2.6.3 Multiple Pattern Search

Performa pencarian pola tunggal dalam sebuah teks dengan algoritma Rabin-Karp tidak sebaik dengan algoritma pencocokan *string* lainnya. Algoritma Rabin-Karp lebih cocok digunakan untuk mencari banyak polaatau *multiple pattern search* (Harris & Ross, 2005). Saat melakukan pencarian pola sebanyak *k* di dalam teks sepanjang *n*, kompleksitas waktu yang dimiliki algoritma ini adalah O(n + k). Dalam kasus yang sama, kompleksitas waktu yang dimiliki algoritma lain adalah O(nk). Dengan melihat kedua kompleksitas waktu, algoritma Rabin-Karp memang lebih capat dalam mencari banyak pola dibandingkan algoritma pencocokan *string* lainnya. Konsep dalam mencari banyak pola sama dengan saat mencari pola tunggal, yaitu menggunakan teknik *hashing* (Firdaus, 2003).

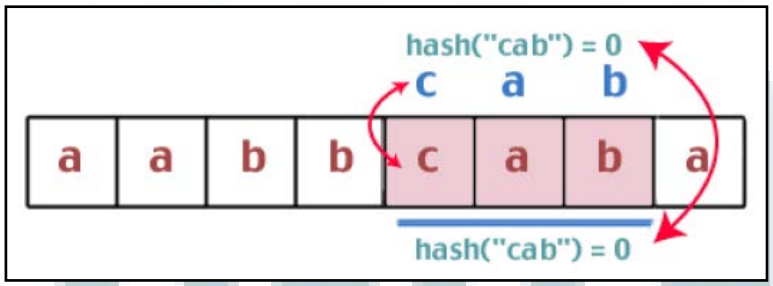
```
function RabinKarpSet(string s[1..n], set of
string subs, m):
    set hsubs := emptySet

foreach sub in subs
        insert hash(sub[1..m]) into hsubs
    hs := hash(s[1..m])

    for i from 1 to n-m+1
if hs ∈ hsubs and s[i..i+m-1] ∈ subs
return i
        hs := hash(s[i+1..i+m])
return not found
```

Gambar 2.7 *Pseudocode Multiple Pattern Search* Algoritma Rabin-Karp (Rabin & Karp, 1987)

2.6.4 Skenario Pencarian Algoritma Rabin-Karp

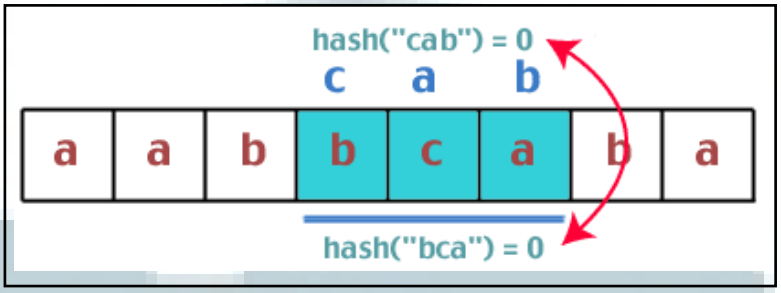
Ketika algoritma Rabin-Karp melakukan pencarian pola dalam suatu teks, terdapat tiga skenario yang bisa terjadi. Berikut adalah penjelasan serta ilustrasi dari ketiga skenario tersebut (Puntambekar, 2009).

1. *Successful hit* → Hasil modulus yang dibandingkan sama dan setiap karakter yang dibandingkan dari teks dan pola cocok.



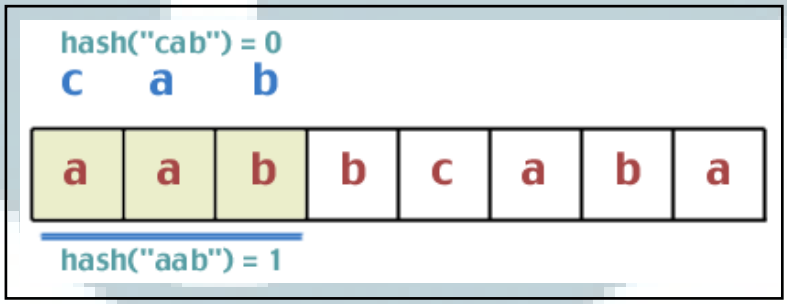
Gambar 2.8 *Successful Hit* (Sparknotes, 2003)

2. *Spurious hit* → Hasil modulus yang dibandingkan sama tetapi terdapat sebuah karakter yang tidak cocok ketika dibandingkan.



Gambar 2.9 *Spurious Hit* (Sparknotes, 2003)

3. *Unsuccessful hit* → Hasil modulus yang dibandingkan tidak sama.



Gambar 2.10 *Unsuccessful Hit* (Sparknotes, 2003)

2.6.5 Algoritma Rabin-Karp Modifikasi

Semakin besar bilangan prima yang dipakai, frekuensi kejadian *spurious hit* dapat dikurangi dan pencocokan ekstra tidak perlu dilakukan lagi (Cormen, Leiserson, Rivest, & Stein, 2003). Namun, penggunaan modulus tidak menutupi kemungkinan terjadinya *spurious hit.* Keberadaan *spurious hit* hanya menjadi beban dalam pencarian pola karena harus melakukan perbandingan berlebih. Oleh karena itu, algoritma Rabin-Karp dimodifikasi kriteria sebagai berikut.

REM(n1/q) = REM(n2/q) QUOTIENT(n1/q) = QUOTIENT(n2/q)

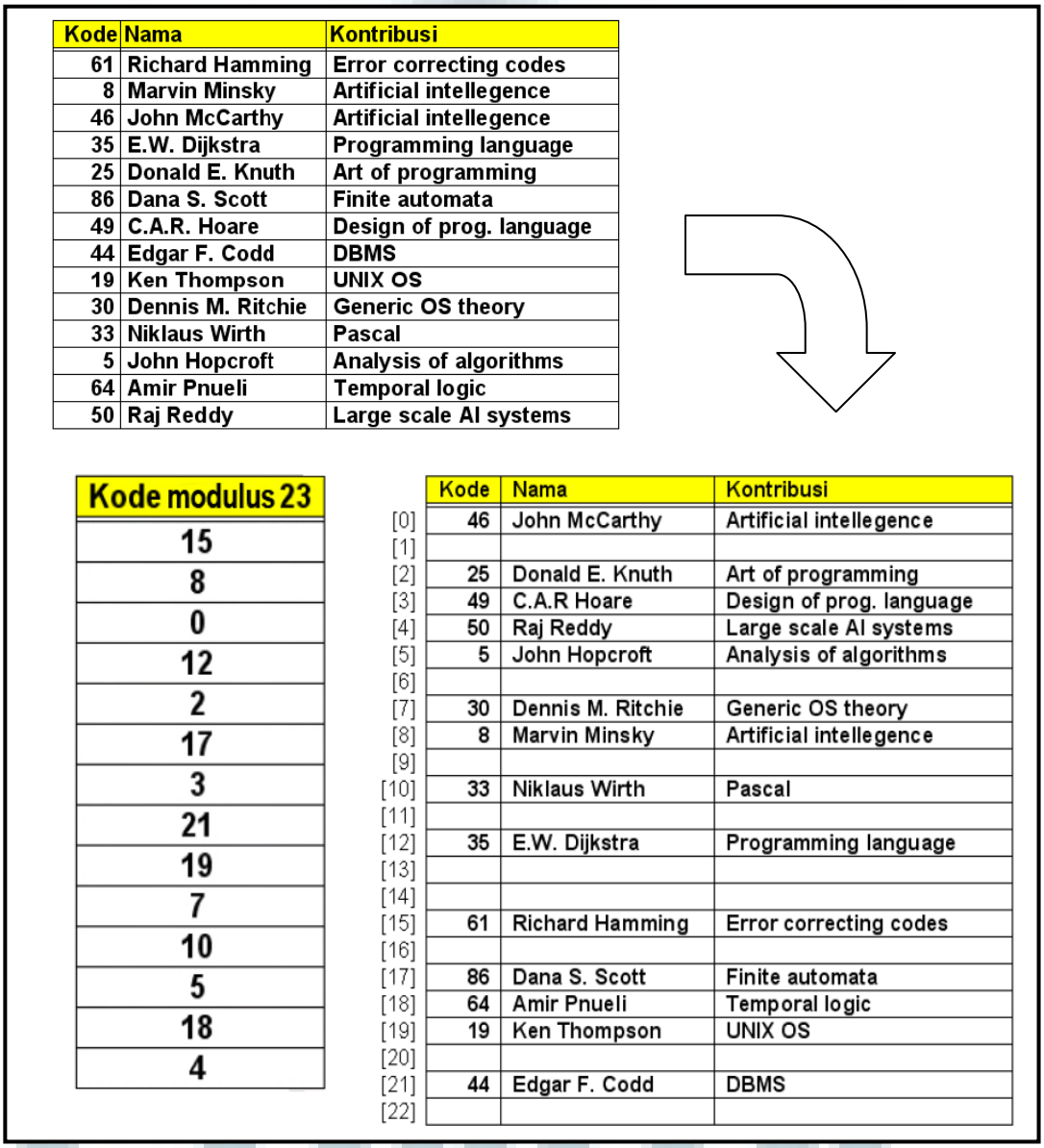
.....Rumus 2.1

Untuk melakukan pencocokan *string* harus memenuhi dua syarat di atas. Pertama, hasil modulus dengan bilangan prima dari kedua nilai *hash* harus sama. Kedua, hasil bagi dengan bilangan prima dari kedua nilai *hash* harus sama. Jika kedua syarat sudah terpenuhi, *successful hit.* Sebaliknya, *unsuccessful hit.* Dengan demikian, tidak ada lagi *spurious hit* (Chillar & Kochar, 2008).

2.7 Hashing

Dalam ilmu komputer, *hashing* memiliki dua definisi. Definisi pertama *hashing* adalah suatu teknik dimana data ditempatkan pada alamat yang cocok dalam suatu tabel yang disebut dengan *hash table* (Puntambekar, 2008). Penentuan alamat penempatan menggunakan sebuah fungsi yang disebut dengan *hash function. Hash function* adalah sebuah algoritma yang mengubah teks dengan ukuran variatif menjadi nilai dengan ukuran yang tetap (*hash value*). Cara kerja *hash function* adalah data diambil satu per satu, kemudian dilakukan beberapa komputasi terhadap data tersebut sehingga dihasilkan suatu nilai *hash* dan dari nilai itu bisa ditentukan alamat penempatan data dalam tabel. Syarat *hash function* yang baik adalah sederhana, prosesnya cepat, dan data terdistribusi merata. Perlu diingat bahwa *hash function* bukanlah meng-*encode* suatu *string* menjadi nilai *hash.* Dengan demikian, tidaklah mudah untuk mengubah nilai *hash* ke suatu *string* karena nilai *hash* memiliki ukuran yang tetap, sedangkan panjang

string masukan bisa bervariatif. Contoh pada gambar 2.11 menggunakan *hash function* yaitu kode dimodulus dengan bilangan 23 (Holden, 2012).



Kode	Nama	Kontribusi
61	Richard Hamming	Error correcting codes
8	Marvin Minsky	Artificial intellegence
46	John McCarthy	Artificial intellegence
35	E.W. Dijkstra	Programming language
25	Donald E. Knuth	Art of programming
86	Dana S. Scott	Finite automata
49	C.A.R. Hoare	Design of prog. language
44	Edgar F. Codd	DBMS
19	Ken Thompson	UNIX OS
30	Dennis M. Ritchie	Generic OS theory
33	Niklaus Wirth	Pascal
5	John Hopcroft	Analysis of algorithms
64	Amir Pnueli	Temporal logic
50	Raj Reddy	Large scale AI systems

Kode modulus 23
15
8
0
12
2
17
3
21
19
7
10
5
18
4

	Kode	Nama	Kontribusi
[0]	46	John McCarthy	Artificial intellegence
[1]			
[2]	25	Donald E. Knuth	Art of programming
[3]	49	C.A.R Hoare	Design of prog. language
[4]	50	Raj Reddy	Large scale AI systems
[5]	5	John Hopcroft	Analysis of algorithms
[6]			
[7]	30	Dennis M. Ritchie	Generic OS theory
[8]	8	Marvin Minsky	Artificial intellegence
[9]			
[10]	33	Niklaus Wirth	Pascal
[11]			
[12]	35	E.W. Dijkstra	Programming language
[13]			
[14]			
[15]	61	Richard Hamming	Error correcting codes
[16]			
[17]	86	Dana S. Scott	Finite automata
[18]	64	Amir Pnueli	Temporal logic
[19]	19	Ken Thompson	UNIX OS
[20]			
[21]	44	Edgar F. Codd	DBMS
[22]			

Gambar 2.11 Penempatan Data ke *Hash Table* (Cormen, 2003)

Definisi *hashing* yang kedua adalah transformasi aritmetika sebuah *string* dari karakter menjadi nilai yang merepresentasikan *string* aslinya. Konsepnya sama seperti sebelumnya, *hash function* dibutuhkan untuk mengubah *string*

menjadi sebuah nilai *hash.* Berikut adalah contoh sederhana dari *hashing* (Kumar, Sofat, Jain, & Aggarwal, 2012).

Tabel 2.1 *String Hashing* (Gittleman, 2011)

String	Panjang*String*	*Hash Value*
text	4	2.087.956.376
print	5	187.024.980
repeat	6	2.123.762.162
address	7	76.895.091

Dalam algoritma Rabin-Karp, umumnya memakai sebuah *hash function* yang sederhana dan hanya menggunakan operasi penjumlahan dan perkalian. Hasil berupa angka yang dihasilkan bisa sangat besar. Untuk menghindari hal tersebut, hasil akan dimodulus oleh sebuah bilangan prima. Berikut adalah rumus dari *hash function* tersebut (Candan & Sapino, 2010).

$$\text{hash}(x) = (x_0 \,.\, 10^{k-1}) + (x_1 \,.\, 10^{k-2}) + \ldots. + (x_k \,.\, 10^0)$$ …Rumus 2.2

x : *String* masukan
k : Panjang *string*
x_k : Nilai ASCII dari karakter ke k

Hashing dapat digunakan dalam pencarian kata dari *database*. Apabila tidak di-*hash*, pencarian akan dilakukan per karakter pada nama yang panjangnya bervariasi dan ada 26 kemungkinan pada setiap karakter. Dengan adanya *hash*, pencarian akan menjadi lebih cepat karena hanya akan membandingkan beberapa digit angka dengan 10 kemungkinan setiap angka. Nilai *hash* pada umumnya

digambarkan sebagai *fingerprint* yaitu suatu *string* pendek yang terdiri atas huruf dan angka yang terlihat acak (Firdaus, 2003).

2.8 N-grams

N-grams merupakan suatu urutan dari banyak *terms* dengan panjang N. Dalam konteks ini, *terms* paling banyak berupa kata. *N-grams* dibentuk dengan mengambil potongan-potongan kata sepanjang N karakter secara kontinu dan tumpang tindih dari awal sampai akhir dari teks. Potongan-potongan kata yang sudah terbentuk bisa dipakai untuk membandingkan beberapa dokumen teks (Náther, 2005).

Selain membentuk *n-grams* dengan mengambil potongan kata secara kontinu dan tumpang tindih (*overlapping n-grams*), *n-grams* juga bisa dibentuk dengan mengambil potongan kata secara kontinu tetapi tidak tumpang tindih (*consecutive n-grams*). Berikut adalah contoh kedua cara pembentukan *n-grams*dari kata “kuliah” dengan ukuran sebesar 2 (Salmela, Tarhio, & Kytojoki, 2006).

Overlapping n-grams : ku, ul, li, ia, ah

Consecutive n-grams : ku, li, ah

Dalam membentuk *n-grams*, harus diketahui terlebih dahulu ukurannya. Berdasarkan ukuran yang sering dipakai, *n-grams* diklasifikasikan menjadi empat jenis dan dijelaskan pada tabel berikut (Lalwani, Bagmar, & Parikh, 2011).

Tabel 2.2 Nama *N-Grams* Berdasarkan Ukuran (Lalwani, 2011)

Ukuran	Nama
1	*Unigrams*
2	*Bigrams* atau *digrams*
3	*Trigrams*
Lebih dari 3	*N-grams*

Teks yang dijiplak bisa saja berbeda dari aslinya. Dengan adanya penggunaan *n-grams* dan perbedaan teks, urutan kata yang sama masih bisa ditemukan. Kemiripan ini dipengaruhi saat ada beberapa bagian teks asli yang memiliki struktur yang sama dengan teks yang dijiplak. Dengan begitu, keduanya memiliki struktur sintaktik yang sama dalam kalimat berturut-turut atau seluruh paragraf (Stamatatos, 2011).

2.9 Nilai Similarity

Cara untuk menghitung *similarity* berdasarkan *n-grams* memiliki dua tahap sederhana. Pertama adalah membentuk *n-grams* dari teks. Kemudian dari *n-grams* yang sudah dibentuk, hitung jumlah kata yang sama dan identik secara struktural. Setelah didapatkan, nilai *similarity* bisa dihitung menggunakan *Dice's coefficient. Dice's coefficient* adalah sebuah statistik yang digunakan untuk menghitung kesamaan antara dua himpunan. Himpunan yang digunakan adalah hasil *n-grams* dari dokumen asli dan dokumen uji. Berikut adalah rumusnya (Kosinov, 2001).

$$S = \frac{2C}{A + B}$$ .....Rumus 2.3

S : Nilai *similarity*
A dan B : Jumlah dari *n-grams* teks pertama dan kedua
C : Jumlah dari *n-grams* yang sama

Semakin banyak jumlah *n-grams* yang sama, semakin besar juga nilai *similarity*. Perlu diingat bahwa anggota yang berulang dalam suatu himpunan hanya dihitung satu kali. Berikut adalah contoh perhitungan nilai *similarity* dari dengan *n-grams* sepanjang 2 (*bigrams*).

Tabel 2.3 Contoh Perhitungan Nilai *Similarity* (Kosinov, 2001)

Kata yang dibandingkan	*Bigrams* yang sama	Nilai *Similarity*
Photography (9) dan *Photographic (10)*	Ph ho ot to og 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

Dalam mendeteksi plagiarisme, persentase nilai *similarity* memiliki lima arti, yaitu (Mutiara & Agustina, 2008) :

1. 0% menandakan kedua dokumen tersebut benar-benar berbeda baik dari segi isi dan kalimat secara keseluruhan.
2. <15% menandakan kedua dokumen tersebut hanya mempunyai sedikit kesamaan.

3. 15-50% menandakan dokumen tersebut termasuk plagiat tingkat sedang.
4. >50% menandakan dokumen tersebut mendekati plagiarisme.
5. 100% menandakan bahwa dokumen tersebut adalah plagiat karena dari awal sampai akhir mempunyai isi yang sama persis.