BAB II

LANDASAN TEORI

2.1 Plagiarisme

Plagiarisme merupakan perbuatan secara sengaja atau tidak sengaja dalam memperoleh atau mencoba memperoleh kredit atau nilai untuk suatu karya ilmiah [3, 10], dengan mengutip sebagian atau seluruh karya ilmiah orang lain, tanpa menyatakan sumber secara tepat dan memadai istilah tersebut sesuai dengan Permendiknas No 17 tahun 2010, Pasal 1 Ayat 1. Sedangkan pendapat Ir. Balza Achmad, M. Sc. E adalah berbuat sesuatu seolah-olah karya orang lain tersebut adalah karya kita dan mengakui hasil karya tersebut milik kita [3, 7]. Menurut Moeliono dalam referensi Mulyana, plagiarisme adalah pengambilan karya orang lain, dan dipublikasikan sebagai karya miliknya. Pada kenyataannya, di dunia ilmiah, pengambilan tulisan atau mengutip karya orang lain tersebut, kadang-kadang dianjurkan namun dengan aturan dan norma yang telah berlaku dan disepakati secara luas di dunia akademik. Persoalannya, seberapa besar kadar pengambilan kutipan, bagaimana cara mengutip, dan apakah pihak pengutip menyertakan sumber kutipannya atau tidak adalah sesuatu yang muskil dan sulit ditelusuri [11]. Di Indonesia perlindungan hak cipta diatur dalam Undang-Undang Republik Indonesia Nomer 19 Tahun 2002 Tentang Hak Cipta. Oleh karena itu, kegiatan plagiarisme atau yang lebih dikenal dengan kata plagiat harus dihindari. Plagiarisme dapat dianggap sebagai tindak pidana karena mencuri hak cipta orang lain. Di dunia pendidikan, pelaku plagiarisme akan mendapat hukuman berat seperti dikeluarkan dari sekolah/universitas. Pelaku plagiarisme disebut sebagai plagiator[3].

Plagiarisme tidak hanya melanggar hak cipta atau kepemilikan tetapi tindakan penipuan dan menimbulkan kesalah pahaman mengenai orisinalitas dari penulis yang sebenarnya. Para siswa/ mahasiswa dan peneliti diperbolehkan untuk menciptakan suatu karya baru yang timbul dari pengembangan ide orang lain. Tetapi pemanfaatan ide orang lain tanpa membubuhkan pernyataan sumber

merupakan tindakan yang tidak dapat diterima. Namun, Mulyana menyatakan bahwa cara mencantumkan relevansi mengarahkan mahasiswa untuk melakukan duplikasi atau plagiarisme. Sadar atau tidak, cara mengutip yang dilakukan telah mendekatkan skripsi mereka pada skripsi orang lain. Dari sinilah antara lain gejala plagiarisme muncul. Dalam buku *Bahasa Indonesia: Sebuah Pengantar Penulisan Ilmiah*, [12] menggolongkan hal-hal berikut sebagai tindakan plagiarisme:

1. Mengakui tulisan orang lain sebagai tulisan sendiri.
2. Mengakui gagasan orang lain sebagai pemikiran sendiri.
3. Mengakui temuan orang lain sebagai kepunyaan sendiri.
4. Mengakui karya kelompok sebagai kepunyaan atau hasil sendiri.
5. Menyajikan tulisan yang sama dalam kesempatan yang berbeda tanpa menyebutkan asal usulnya.
6. Meringkas dan memparafrasekan (mengutip tak langsung) tanpa menyebutkan sumbernya.
7. Meringkas dan memparafrasekan dengan menyebut sumbernya, tetapi rangkaian kalimat dan pilihan katanya masih terlalu sama dengan sumbernya.

2.1.1 Kategori Plagiarisme

Kategori praktek plagiat [13-15] berdasarkan cara yang digunakan, diantaranya**:**

1. *Copy&Paste plagiarism*, menyalin setiap kata tanpa perubahan.
2. *Disguised plagiarism*, tergolong kedalam praktek menutupi bagian yang disalin, teridentifikasi ke dalam empat teknik, yaitu shake&paste, expansive plagiarism, contractive plagiarism, dan mosaic plagiarism.
3. *Technical disguise*, teknik meringkas untuk menyembunyikan konten plagiat dari deteksi otomatis dengan memanfaatkan kelemahan dari metode analisis teks dasar, misal dengan mengganti huruf dengan simbol huruf asing.
4. *Undue paraphrasing*, sengaja menuliskan ulang pemikiran asing dengan pemilihan kata dan gaya plagiator dengan menyembunyikan sumber asli.

5. *Translated plagiarism,* mengkonversi konten dari satu bahasa ke bahasa lain.
6. *Idea plagiarism*, menggunakan ide asing tanpa menyatakan sumber.

2.1.2 Metode Pendeteksi Plagiarisme

Metode pendeteksi plagiarisme dibagi menjadi tiga metode yaitu metode Fingerprint, metode *String-Matching*, dan metode *Tree-Matching* [15, 16]. Berikut ini penjelasan dari masing-masing metode pendeteksi plagiarisme:

1. Metode *Fingerprinting* merupakan metode yang digunakan untuk mendeteksi beberapa atribut dan struktur dari dokumen. Atribut tersebut diantaranya jumlah kata per baris, jumlah kata unik, dan jumlah kutipan pendek. Contoh sistem yang menggunakan metode *fingerprinting* yaitu *MOSS* dan *Accuse*.
2. Metode *String-Matching* atau pencocokkan string adalah membandingkan string dokumen dengan string dokumen lain dan dihitung kesamaannya. Contoh sistem yang menggunakan metode pencocokkan string yaitu *YAP3* dan *Jplag*.
3. Metode *Tree-Matching* Metode *Tree-Matching* adalah dokumen yang dibandingkan harus memiliki aturan struktur yang sama. Contoh sistem yang menggunakan metode *Tree-Matching* yaitu *SIM* dan *Brass*.

2.2 Algoritma *String Matching*

Menurut adam beberapa macam algoritma yang digunakan dalam metode pencocokkan string diantaranya

1. Algoritma *Brute Force*

   Algoritma *Brute Force* dalam penerapan pencocokkan string didefinisikan sebagai berikut: Jika diasumsikan teks adalah sebuah table karakter maka pattern yang juga merupakan table karakter dengan panjang lebih kecil dari teks, maka Algoritma Brute Force diimplementasikan sebagai berikut :

   1. Mula-mula mencocokkan pattern diawal teks.

2. Dengan bergerak dari kiri ke kanan bandingkan setiap karakter di Dalam pattern dengan karakter yang sesuai dengan teks sampai
3. Semua karakter yang dibandingkan sesuai dengan pattern yang dicari.
4. Dijumpai sebuah ketidakcocokkan karakter.
5. Bila dijumpai ketidak cocokkan karakter pada pattern dan teks belum berakhir geser pattern satu karakter ke kanan.

2. *Knuth Morris Pratt*

   Pada Algoritma ini memelihara informasi yang digunakan untuk melakukan sejumlah pergeseran. Algoritma ini menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter seperti pada Algoritma *Brute Force*.

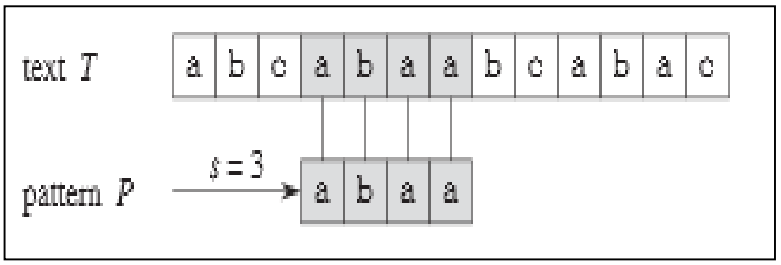
3. Algoritma *Boyer-Moore*

   Algoritma ini akan bertambah cepat jika kata yang dicari panjang. Setiap pencocokkan yang gagal antara teks dan kata yang dicari, Algoritma ini menggunakan informasi yang didapat dari proses awal untuk melewati karakter-karakter yang tidak cocok.

4. Algoritma *Rabin-Karp*

   Algoritma ini adalah Algoritma acak sederhana yang cenderung berjalan dalam waktu linier. Pada penelitian (Adam et al., 2009), Algoritma *Rabin-Karp* adalah pilihan tepat 10 untuk pencocokkan dengan banyak pola (Adam et al., 2009). Kasus terburuk dari Algoritma ini seperti pada Algoritma *Brute Force*. Semua Algoritma yang akan dibahas mengeluarkan semua kehadiran pola dalam teks. Sebuah contoh pada Gambar menunjukan pola/ pattern P = abaa dalam text T = abcabaabcabac. Pola ini hanya terjadi satu kali pergeseran(s) dalam setiap teks. Nilai s = 3 karena melakukan pergeseran sebanyak 3 kali. Sebuah garis vertikal yang menghubungkan setiap string dari pola untuk pencocokkan string dalam teks (Cormen et al.,

2009). Algoritma string matching yang digunakan dalam tugas akhir ini adalah Algoritma Rabin-Karp.



Gambar 2. 1 Contoh String Matching

2.3 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[17]. 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). Langkah-langkah dalam Algoritma *Rabin-Karp*:

1. Menghilangkan tanda baca dan mengubah ke teks sumber dan kata yang ingin dicari menjadi kata-kata tanpa huruf.
2. Membagi teks kedalam gram-gram yang ditentukan nilai *K-Gram*nya.

3. Mencari nilai hash dengan fungsi hash dari tiap kata yang terbentuk.
4. Mencari nilai hash yang sama antara dua teks.

2.3.1 Konsep Algoritma *Rabin-Karp*

Berikut adalah karakteristik utama dari Algoritma *Rabin-Karp* Menggunakan fungsi *Hashing*.

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

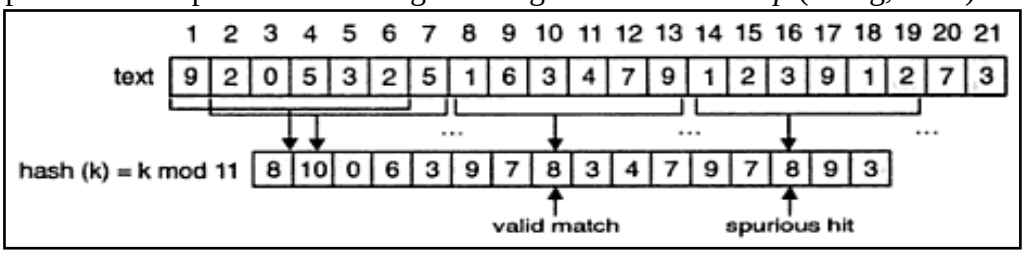
Secara garis besar, Algoritma *Rabin-Karp* dapat dijelaskan dengan *pseudocode* berikut.

```
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. 2 Pseudocode Algoritma Rabin

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).

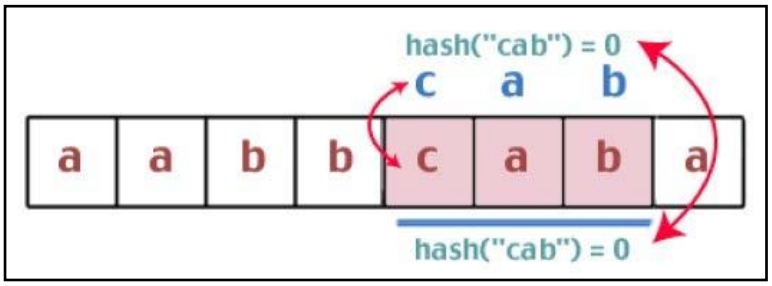


Gambar 2. 3 Ilustrasi Algoritma Rabin karp

2.3.2 Skenario Pencarian Algoritma *Rabin-Karp*

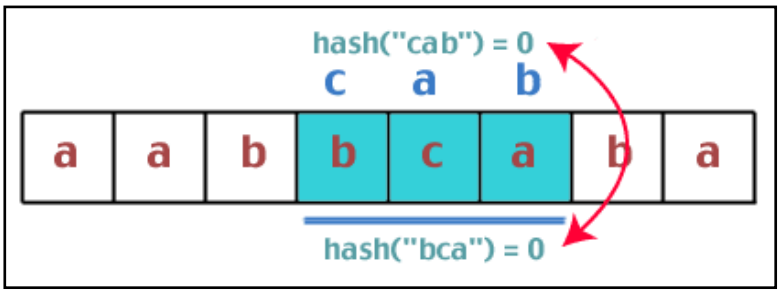
Ketika Algoritma *Rabin-Karp* melakukan pencarian pola dalam suatu teks, terdapat tiga skenario yang bisa terjadi. Berikut adalah penjelasan serta ilustrasi dari ketiga skenario tersebut (Puntambekar, 2009).

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



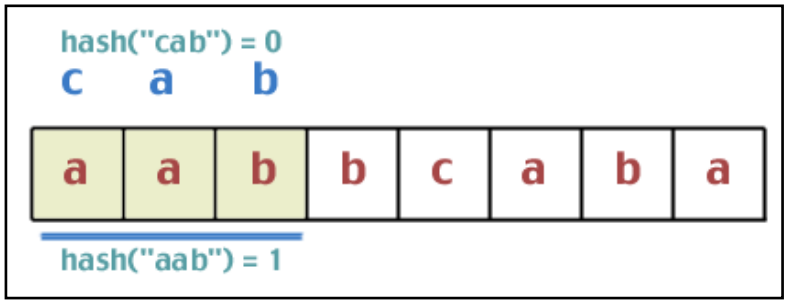
Gambar 2. 4 Successful Hit (Sparknotes, 2003)

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



Gambar 2. 5 Spurious Hit (Sparknotes, 2003)

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



Gambar 2. 6 Unsuccessful Hit (Sparknotes, 2003)

2.3.3 *Algoritma Rabin-Karp Modification*

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).

2.4 Cara Kerja Algoritma *Rabin-Karp*

Berikut ini merupakan cara kerja Algoritma *Rabin-Karp* yang dapat dijelaskan yaitu:

1. Input teks/upload yaitu memasukkan teks uji dan teks penguji. dokumen uji sebagai teks atau dokumen yang akan diuji kemiripannya berdasarkan text asli. Pembacaan isi text yang diproses terdiri dari huruf dan angka.
2. Normalisasi input text adalah penghapusan simbol dan text non-alphanumeric atau menghilangkan tanda baca, merubah huruf kehuruf kecil dan menghilangkan tanda baca.
3. *Tokenizing* berdasarkan *K-Gram* merupakan penghilangan kata dan tanda baca yang dianggap kurang penting seperti kata penghubung (yang, dan, di, dll) serta karakter tanda titik, koma dan sebagainya. Selain itu pada tahapan ini string yang diolah disamaratakan menjadi *lower case* seluruhnya.
4. Menghitung *Hashing* merupakan proses mengubah karakter menjadi bilangan hash.
5. *Text matching* berdasarkan nilai *Hashing* untuk mencocokkan nilai hash satu persatu
6. Perhitungan *Similarity*

   Dengan menggunakan rumus dibawah ini:

   S = (IH / THA + THB) x 100%

   Dimana:

   S = *Similarity* (Kesamaan)

IH = jumlah pola yang sama

THA = jumlah *Hashing* dokumen uji

THB = jumlah hasing dokumen penguji

2.5 Tahap Text *Preprocessing*

Tahap ini melakukan analisis semantik (kebenaran arti) dan sintaktik (kebenaran susunan) terhadap teks. Tujuan dari pemrosesan awal adalah untuk mempersiapkan teks menjadi data yang akan mengalami pengolahan lebih lanjut. II-8 Operasi yang dapat dilakukan pada tahap ini meliputi proses untuk menghilangkan bagian-bagianyang tidak diperlukan atau pembersihan teks yang dilakukan untuk mengubah data-data berkulitas yaitu data yang telah memenuhi persyaratan untuk dieksekusi padasebuah Algoritma. Bentuk pembersihan teks ini seperti menghilangkan spasi, tanda baca, mengubah huruf kapital menjadi huruf kecil dan menghilangkan karakter-karakter yang tidak relevan lainnya.

2.5.1 *Tokenizing*

Tokenizing adalah proses penghilangan tanda baca pada kalimat yang ada dalam dokumen sehingga menghasilkan kata-kata yang berdiri sendiri- sendiri.

2.5.2 *Filtering*

Filtering adalah tingkatan untuk mengambil kata penting dengan menggunakan algoritma *stoplis*t (menghapus kata-kata yang kurang penting) atau daftar kata (berisi kata-kata penting). *Stoplist/stopword* adalah kata non-penjelas yang dapat membuang . Contoh stopword adalah "*the", "and", "at", "from*", dan seterusnya.

2.5.3 *Stemming*

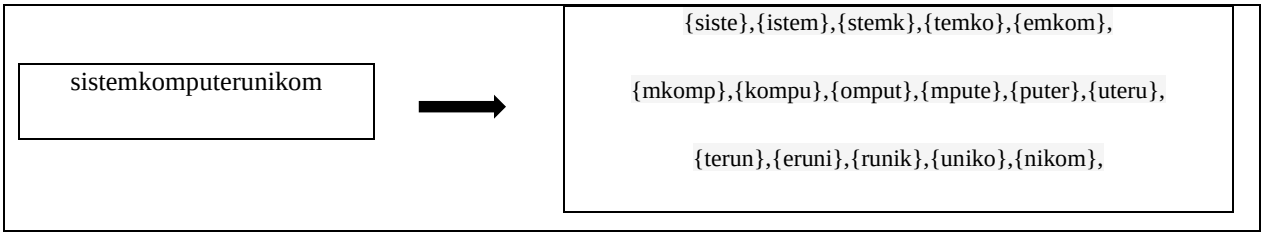
Stemming adalah proses mengubah kata menjadi kata dasarnya dengan menghilangkan imbuhan-imbuhan pada kata dalam dokumen atau mengubah kata kerja menjadi kata benda. Stem (akar kata) adalah bagian dari kata yang tersisa setelah dihilangkan imbuhannya (awalan dan akhiran).

2.5.4 *Parsing K-Gram*

Parsing K-Gram adalah membentuk pola kata pada teks dengan memecah kata menjadi potongan-potongan dimana setiap potongan mengandung karakter sebanyak k. *K-Gram* merupakan sebuah metode yang diaplikasikan untuk pembangkitan kata atau karakter. Metode *K-Gram*s ini digunakan untuk mengambil potongan-potongan karakter huruf sejumlah k dari sebuah kata yang secarakontinuitas dibaca dari teks sumber hingga akhir dari dokumen.

Dalam Markov Model nilai *K-Gram* yang sering digunakan yaitu, 2-gram (bigram), 3-gram (trigram), 4-gram, 5-gram dan seterusnya). Dalam natural language processing, penggunaan *K-Gram*, 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 kemiripan, 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 banyak kesamaan kata dalam pemrosesan nilai kemiripan. Namun jika menggunakan 3 *K-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.

Contoh menggunakan [12, 18] nilai 5 *K-Gram* yang mana semakin mengecilkan kemungkinan terjadinya kesamaan kata karena semakin besar *K-Gram* semakin kecil adanya kesamaan kata.



Gambar 2. 7 Contoh Parsing 5 K-Gram

2.5.5 *Hashing*

Hashing adalah suatu cara untuk mentransformasi sebuah string menjadi suatu nilai yang unik dengan panjang tertentu *(fixed-length)* yang berfungsi sebagai penanda (signature) string tersebut. Panjang string sesuai dengan nilai k-gram yang ditentukan. Hash function atau fungsi hash adalah suatu cara menciptakan fingerprint dari berbagai data masukkan. Hash function akan mengganti atau mentransformasikan data tersebut untuk menciptakan fingerprint, yang biasa disebut hash value.

Nilai hash yang akan dicari dengan fungsi hash dalam algoritma Rabin-Karp merupakan representasi dari nilai *ASCII* (American Standar Code for Information Interchange) yang menempatkan angka numerik pada karakter, angka, tanda baca dan karakter-karakter lainnya. *ASCII* menyediakan 256 kode yang dibagi ke dalam dua himpunan standar dan diperluas yang masing-masing terdiri dari 128 karakter. Himpunan ini merepresentasikan total kombinasi dari 7 atau 8 bit, yang kemudian menjadi angka dari bit dalam 1 byte. *ASCII* standar menggunakan 7 bit untuk tiap kode dan menghasilkan 128 kode karakter dari 0 sampai 127 (heksadesimal 00H sampai 7FH). Himpunan *ASCII* yang diperluas menggunakan 8 bit untuk tiap kode dan menghasilkan 128 kode tambahan dari 128 sampai 255 (heksadesimal 80H sampai FFH) [19].

Rolling hash adalah sebuah fungsi hash yang input-nya dikelompokkan ke dalam suatu blok yang digerakkan melewati input secara keseluruhan. Beberapa fungsi hash memungkinkan rolling hash untuk dikomputasi dengan cepat. Nilai hash yang baru dapat dengan cepat dihitung dari nilai hash yang lama dengan cara menghilangkan nilai lama dari kelompok hash dan menambahkan nilai baru ke dalam kelompok tersebut [5].

Kunci dari performa algoritma Rabin-Karp adalah komputasi yang efektif dari nilai hash dari substring-substring yang berurutan pada teks. Algoritma Rabin-Karp melakukan perhitungan nilai hash dengan memperlakukan setiap substring sebagai sebuah angka dengan basis tertentu (Adam et al., 2009). Dengan begitu tidak perlu melakukan iterasi dari indeks pertama sampai terakhir. Hal ini tentu menghemat nilai hash dari sebuah string [5].

$$H(C1\,..\,Ck\,) = c1 * b\,(k-1) + c2 * b\,(k-2) + \ldots\,.\,. + c(k-1) * b\;k + ck$$

$$H(C2\ldots Ck{+}1) = (H(C1\,..\,Ck) - c1 * b\,(k-1)\,) * b + c(k{+}1)$$

Keterangan :

H : substring

c : nilai *ASCII* per-karakter

b : basis k : banyak karakter

Sebagai contoh, jika substring yang ingin dicari adalah “machio”. Karakter (k) yang ditentukan sebanyak 5 sehingga substring machio terbagi menjadi 2 yaitu “machi” dan achio. Sedangkan basis (b) yang digunakan adalah 3, rumus perhitungan nilai hash:

$H(machi) = ascii(m) * 3\,(4) + ascii(a) * 3\,(3) + ascii(c) * 3\,(2) + ascii(h) * 3\,(1) + ascii(i) * 3\,(0)$

$H(machi) = 109 * 81 + 97 * 27 + 99 * 9 + 104 * 3 + 105 * 1 = 12756$

Nilai hashing pada substring “machi” bernilai 12756. Untuk substring selanjutnya sebagai contoh, “achio” k = 5 dan b = 3.

$H(achio) = (H(machi) - ascii(m) * 3\,(4)\,) * 3 + ascii\,(o)$

$H(achio) = (12756 - 109 * 81) * 3 + 111 = 11892$

Nilai hashing pada substring “achio” bernilai 11892.

2.5.6 *Similarity*

Mengukur *Similarity* (kemiripan) dan jarak antara dua entitas informasi adalah syarat inti pada semua kasus penemuan informasi, seperti pada *Information Retrieval* dan *Data Mining* yang kemudian dikembangkan dalam bentuk aplikasi, salah satunya adalah sistem deteksi plagiat. Penggunaan ukuran *Similarity* yang tepat tidak hanya meningkatkan kualitas pilihan informasi tetapi juga membantu mengurangi waktu dan biaya proses menyarankan untuk mengaplikasikan *Dice's Similarity Coefficient* dalam penghitungan nilai *Similarity* yang menggunakan pendekatan *k- gram.*

S = (IH / THA + THB) x 100%

Dimana:

S = *Similarity* (Kesamaan)

IH = jumlah pola yang sama

THA = jumlah *hashing* dokumen uji

THB = jumlah *hasing* dokumen penguji

Tabel 2. 1 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

2.5.7 Persentase Kemiripan

Dalam penelitian (Mutiara, 2011), *range* persentase nilai kemiripan yang digunakan adalah:

1. 0% : Hasil uji 0% berarti kedua dokumen tersebut benar-benar berbeda baik dari segi isi dan kata secara keseluruhan.
2. < 15% : Hasil uji 15% berarti kedua dokumen tersebut hanya mempunyai sedikit kesamaan.
3. 15 - 50% : Hasil uji 15-50% berarti menandakan dokumen tersebut termasuk kemiripan tingkat sedang.
4. > 50% : Hasil uji lebih dari 50% berarti dapat dikatakan bahwa dokumen tersebut mendekati kemiripan.
5. 100% : Hasil uji 100% menandakan bahwa dokumen tersebut adalah memiliki kemiripan yang sama karena dari awal sampai akhir mempunyai isi yang sama persis.

2.6 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.

2.7 Software Pendukung

Software pendukung merupakan alat yang digunakan untuk membangun sistem atau aplikasi yang digunakan. Software pendukung yang digunakan yaitu:

2.7.1 NoSQL

NoSQL merupakan makna dari “Not Only SQL” merupakan database jenis non-relasional yang menjadi alternatif dari database SQL yang merupakan database relasional, Perbedaan dari kedua jenis database tersebut terdapat di bentuk skema, SQL memiliki skema yang kaku sedangkan database NoSQL memiliki bentuk skema yang lebih fleksibel dan mudah diubah tanpa menggangu sistem yang sedang berjalan.

2.7.2 Java Script

Bahasa pemrograman yang bersifat client side yang permrosesanya dilakukan oleh client sering digunakan pada web browser untuk menciptakan halaman web yang menarik. Menurut Kadir dan Triwahyuni (2013:325) “JavaScript adalah bahasa pemrograman yang biasa diletakkan bersama kode HTML untuk menentukan suatu tindakan”. Sedangkan Menurut Sibero (2013:150) “Javascript adalah bahasa skrip (Scripting language), yaitu kumpulan intruksi perintah yang digunakan untuk mengendalikan beberapa bagian dari sistem operasi”. Berdasarkan pendapat yang dikemukanan diatas dapat disimpulkan bahwa, JavaScript adalah Bahasa pemrograman atau bahasa

skrip yang berisi kumpulan intruksi perintah yang dilletakkan bersama kode HTML.

2.7.3 *Sequence diagram*

Diagram yang menunjukkan kolaborasi dinamis antara beberapa objek. Penggunaannya untuk menampilkan rangkaian pesan yang dikirim antar objek dan interaksi antar objek. Sesuatu yang terjadi di beberapa titik dalam eksekusi sistem.Activity Diagram.

2.7.4 *Visual Studio Code*

Visual Studio Code adalah *Sofware* yang sangat ringan, namun kuat editor kode sumbernya yang berjalan dari desktop. Muncul dengan built-in dukungan untuk *JavaScript*, naskah dan *Node.js* dan memiliki array beragam ekstensi yang tersedia untuk bahasa lain, termasuk *C ++*, *C #* , *Python*, dan *PHP*.