DETEKSI PLAGIAT DOKUMEN MENGGUNAKAN ALGORITMA RABIN-KARP

Hari Bagus Firdaus

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
Jalan Ganesha 10, Bandung
e-mail: if16044@students.if.itb.ac.id

ABSTRAK

Plagiat atau biasa disebut penjiplakan adalah sebuah masalah yang cukup signifikan pada banyak sekolah dan universitas. Hal plagiat yang biasanya dilakukan terhadap konten digital adalah melakukan *copy-paste*, *quote*, dan revisi terhadap dokumen asli. Untuk mengantisipasinya, dibutuhkan suatu cara yang dapat menganalisis teknik-teknik plagiat yang dilakukan. Ada beberapa pendekatan yang bisa diambil, salah satunya dengan mempergunakan algoritma pencarian string Rabin-Karp. Makalah ini hanya membahas secara skematis bagaimana algoritma Rabin-Karp bekerja dalam mendeteksi plagiat pada suatu dokumen, bukan implementasinya dalam sebuah program atau aplikasi.

Kata kunci: Plagiat, Rabin-Karp, pencarian string.

1. PENDAHULUAN

Dokumen digital memang dapat dijiplak dengan sangat mudah. Praktik plagiat atau penjiplakan ini sudah sering terjadi khususnya pada kalangan akademisi baik lingkungan sekolah maupun perguruan tinggi. Tindakan plagiat yang dilakukan siswa atau mahasiswa ini sangat tidak mencerminkan sikap kreatif dan terpelajar sebagai kaum intelektual. Bentuk plagiat ini dapat bermacam-macam seperti misalnya yang paling sering dilakukan yaitu *copy-paste-edit* suatu dokumen baik merupakan pekerjaan teman maupun dokumen yang berasal dari *website*, sehingga menjadi suatu hasil yang berbeda dari sebelumnya. Selain itu plagiat dapat pula ditemukan dalam bentuk *quote* atau kutipan pada sebuah dokumen.

Dalam mendeteksi plagiat secara manual [1], hal terpenting yang harus dilakukan adalah mengetahui bodi inti dari suatu dokumen. Metode berikutnya adalah membuat katalog *paperwork* dan dokumen-dokumen terdahulu. Hal ini biasanya sudah dilakukan pada institusi besar terhadap hasil kerja para mahasiswanya. Selain itu seorang dosen misalnya, dapat pula melakukan pengecekan terhadap gaya penulisan dokumen mahasiswanya. Akan tetapi, seluruh jenis deteksi manual memiliki masalah defisiensi yang cukup serius. Sangat tidak mungkin untuk memeriksa sebuah dikumen dengan ribuan dokumen lainnya yang terdapat dalam katalog, apalagi menafsirkan gaya penulisannya.

Penggunaan *search engine* dapat membantu melakukan pencarian, yaitu dengan memasukkan kata kunci tema dokumen dan biarkan *search engine* menemukan dokumen yang dijiplak yang cocok. Ini memang berguna apabila keseluruhan dokumen yang dijiplak. Namun kurang efektif apabila plagiator hanya menggunakan sebagian dari artikel atau menggabungkan beberpapa pecahan artikel. Selain itu cara ini juga banyak mengkonsumsi waktu.

Metode terakhir adalah dengan cara melakukan pembandingan dengan sumber dokumen asli. Sejumlah dokumen yang dapat dijadikan sumber dokumen asli dapat merupakan *input* secara manual maupun dokumen yang tersebar lewat internet. Pada umumnya, pada metode inilah algoritma Rabin-Karp dan variasinya diterapkan untuk mendapatkan hasil yang maksimal. Algoritma ini biasanya diimplementasikan pada sebuah aplikasi baik yang berbasis *desktop* maupun yang berbasis *web*.

Algoritma Rabin-Karp adalah suatu algoritma pencarian string yang ditemukan oleh Michael Rabin dan Richard Karp. Algoritma ini menggunakan *hashing* untuk menemukan sebuah *substring* dalam sebuah teks. *Hashing* adalah metode yang menggunakan fungsi *hash* untuk mengubah suatu jenis data menjadi beberapa bilangan bulat sederhana. Disebut algoritma "pencarian string" dan bukan "pencocokan string" seperti Knuth-Morris-Pratt atau Boyer-Moore karena memang algoritma Rabin-Karp tidak bertujuan menemukan string yang cocok dengan string masukan, melainkan menemukan pola (pattern) yang sekiranya sesuai dengan teks masukan.

2. ALGORITMA RABIN-KARP

2.1 *Hashing*

Hashing adalah suatu cara untuk mentransformasi sebuah string menjadi suatu nilai yang unik dengan panjang tertentu (*fixed-length*) yang berfungsi sebagai

penanda string tersebut. Fungsi untuk menghasilkan nilai ini disebut fungsi *hash*, sedangkan nilai yang dihasilkan disebut nilai *hash*. Contoh sederhana *hashing* adalah:

```
Firdaus, Hari        Munir, Rinaldi
   Rabin, Michael        Karp, Richard
```

menjadi

```
7864 Firdaus,Hari   9802 Munir, Rinaldi
   1990 Rabin, Michael   8822 Karp, Richard
```

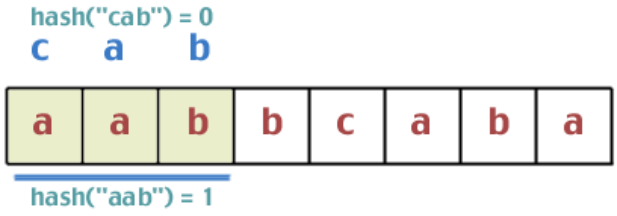
Contoh diatas adalah pengunaan *hashing* dalam pencarian pada *database*. Apabila tidak di-*hash*, pencarian akan dilakukan karakter per karakter pada nama-nama yang panjangnya bervariasi dan ada 26 kemungkinan pada setiap karakter. Namun pencarian akan menjadi lebih mangkus setelah di-*hash* karena hanya akan membandingkan empat digit angka dengan cuma 10 kemungkinan setiap angka.

Nilai *hash* pada umumnya digambarkan sebagai *fingerprint* yaitu suatu string pendek yang terdiri atas huruf dan angka yang terlihat acak (data biner yang ditulis dalam heksadesimal).

2.2 Prinsip Algoritma Rabin-Karp

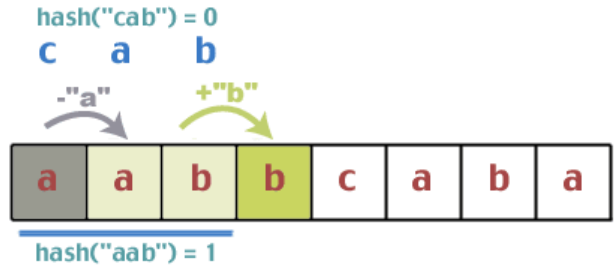
Pada dasarnya, algoritma Rabin-Karp akan membandingkan nilai *hash* dari string masukan dan substring pada teks. Apabila sama, maka akan dilakukan perbandingan sekali lagi terhadap karakter-karakternya. Apabila tidak sama, maka substring akan bergeser ke kanan. Kunci utama performa algoritma ini adalah perhitungan yang efisien terhadapa nilai *hash* substring pada saat penggeseran dilakukan. Berikut dijelaskan contoh cara kerja algoritma Rabin-Karp [2].

Diberikan masukan “cab” dan teks “aabbcaba”. Fungsi *hash* yang dipakai misalnya akan menambahkan nilai keterurutan setiap huruf dalam alfabet (a = 1, b = 2, dst.) dan melakukan modulo dengan 3. Didapatkan nilai *hash* dari “cab” adalah 0 dan tiga karakter pertama pada teks yaitu “aab” adalah 1.



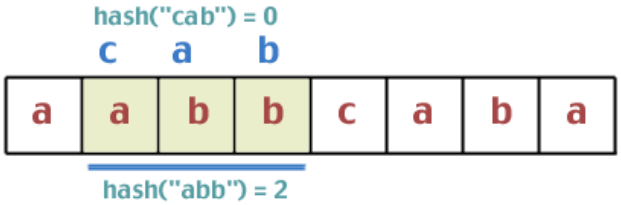
Gambar 1. *Fingerprint* awal

Hasil perbandingan ternyata tidak sama, maka substring pada teks akan begeser satu karakter ke kanan. Algoritma tidak menghitung kembali nilai *hash* substring. Disinilah dilakukan apa yang disebut *rolling hash* yaitu mengurangi nilai karakter yang keluar dan menambahkan nilai karakter yang masuk sehingga didapatkan kompleksitas waktu yang relatif konstan pada setiap kali pergeseran.



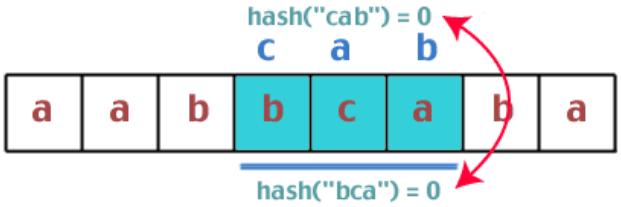
Gambar 2. Menggeser *fingerprint*

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



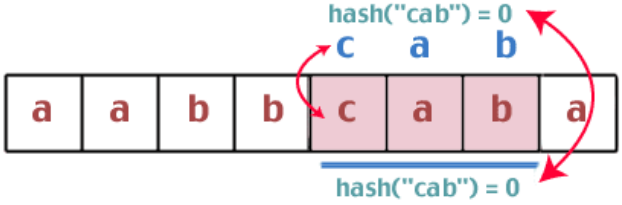
Gambar 3. Pembandingan kedua

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



Gambar 4. Perbandingan keempat (nilai *hash* sama)

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



Gambar 5. Perbandingan kelima (string ditemukan)

Pada perbandingan yang kelima, kedua nilai *hash* dan karakter pembentuk string sesuai, sehingga solusi ditemukan. Dari hasil perhitungan, kompleksitas waktu yang dibutuhkan adalah $O(m+n)$ dengan m adalah panjang string masukan dan n adalah jumlah *looping* yang dilakukan untuk menemukan solusi. Hasil ini jauh lebih mangkus daripada kompleksitas waktu yang didapat menggunakan algoritma *brute-force* yaitu $O(mn)$.

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

```
function RabinKarp (input s:
string[1..m], teks: string[1..n]) → boolean
{
 Melakukan pencarian string s pada
 string teks dengan algoritma Rabin-Karp
}

Deklarasi
   i : integer
   ketemu = boolean

Algoritma:
   ketemu ← false
   hs ← hash(s[1..m])
   for i ← 1 to n-m+1 do
      hsub ← hash(teks[1..i+m-1])
      if hsub = hs then
         if teks[i..i+m-1] = s then
            ketemu ← true
      else
         hsub ← hash(teks[i+1..i+m])
   endfor
   return ketemu
```

Pada *pseudocode* diatas, `teks[i+1..i+m]` adalah substring yang akan dibandingkan dengan string masukan hasil pergeseran sebuah karakter. Pada posisi inilah diselipkan proses *rolling hash.* Fungsi *rolling hash* yang sesuai dengan contoh dan *pseudocode* diatas adalah:

```
teks[i+1..i+m] = teks[i..i+m-1] –
                 teks[i] + teks[i+m]
```

Algoritma Rabin-Karp ternyata masih kurang optimal dan cepat pada pencarian pola string tunggal (*single pattern search*) apabila dibandingkan dengan algoritma Boyer-Moore ataupun algoritma Knuth-Morris-Pratt, tetapi menjadi pilihan bila digunakan untuk mencari string dengan pola yang banyak (*multiple pattern search*) [5].

Bila pola yang ingin ditemukan memiliki panjang, sebut saja k, dan k bernilai besar (yang berarti string masukan panjang dan berpola banyak), algoritma Rabin-Karp dapat disesuaikan dengan tambahan penggunaan *filter* atau *set data structure*, untuk mengecek apakah hasil *hashing* string masukan ada pada kumpulan nilai *hash* dari pola yang dicari. *Filter* digunakan untuk mengeliminasi tanda baca (*punctuation*) dan beberapa kata dan kata sambung yang kurang signifikan untuk diberikan nilai *hash*, sedangkan *set data structure* adalah sekumpulan struktur data yang digunakan untuk membantu pencarian.

Secara garis besar, *pseudocode* untuk algoritma Rabin-Karp untuk pencarian kumpulan string berpola banyak adalah: (diasumsikan semua string masukan pada himpunan *s* memiliki panjang yang sama *m*)

```
function RabinKarpSet (input teks:
string[1..n], s: set of string , m:
integer) → integer
{
 Melakukan pencarian himpunan string s
 yang memiliki panjang m pada string
 teks dengan algoritma Rabin-Karp
 Mengembalikan banyaknya hasil pencarian
 string pada s yang sesuai teks
}

Deklarasi
   i : integer
   str : string
   ketemu = integer

Algoritma:
   ketemu ← 0
   set hs ← (set kosong)
   for each str in s do
      masukkan hash(s[1..m]) kedalam hs
   for i ← 1 to n-m+1
      hsub ← hash(teks[i..i+m-1])
      if hsub ϵ hs then
         if teks[i..i+m-1] = sebuah
         substring dengan hash hsub then
            ketemu ← ketemu + 1
      else
         hsub ← hash(teks[i+1..i+m])
   endfor
   return ketemu
```

Bila algoritma lain dapat mencari string berpola tunggal dalam waktu $O(n)$, jika digunakan untuk mencari pola sebanyak k, maka akan membutuhkan waktu selama $O(nk)$. Sedangkan varian Rabin-Karp diatas lebih mangkus karena diharapkan dapat mencari dengan kompleksitas waktu $O(n+k)$.

3. DETEKSI PLAGIAT DOKUMEN

Algoritma Rabin-Karp digunakan dalam mendeteksi plagiat sebab memungkinkan untuk mencari pola tulisan yang didapat dari substring-substring pada sebuah teks dalam dokumen, dimana algoritma pencarian string tunggal sangat tidak efisien dan praktis. Yang digunakan

tentunya adalah varian algoritma Rabin-Karp untuk pencarian berpola banyak.

Contoh perbandingan sebuah dokumen mahasiswa yang merupakan hasil plagiat dengan dokumen aslinya dari suatu *website* [1].

Tabel 1 Dokumen mahasiswa hasil plagiat dan dokumen asli dari *website*

Dokumen mahasiswa
Even if none of these things happen to you, youd be surprised at what you might find lurking on your computer!
What are malware, adware, and spyware? Malware is a short for malicious software and is a catch all phrase to describe any software that is designed to damage a computer.
Dokumen asli dari http://www.jellico.com/spyware.html
Even if none of these things happen to you, you'ld be surprised at what you might find lurking on your computer!
What is it? Malware (short for "malicious software") is any software developed for the purpose of doing harm to a computer.

Dari tabel 1, sekilas terlihat bahwa mahasiswa tidak melakukan plagiat. Namun, bagi yang mengerti cara kerja algoritma Rabin-Karp dalam mencari string berpola banyak, hal plagiat dapat dengan jelas diketahui.

Pertama-tama adalah melakukan *filtering* dengan menghilangkan beberapa tanda baca yang tidak penting. Proses ini juga dilakukan terhadap dokumen asli. Berikut contoh *filtering* (hanya diambil sebagian kecil dari keseluruhan teks).

```
Even if none of → evenifnone
  What is it?  →  whatisit
```

Dari hasil *filtering*, kata-kata yang akan dijadikan string masukan diambil dengan melakukan pemisahan sebanyak *k*, misalnya, dan dicari seluruh kemungkinan yang mungkin dibentuk dari setiap kata-kata, sehingga didapat pecahan kata-kata. Hal ini disebut dengan *k-gram* [3]. Berikut contoh *k-gram* dengan *k* = 4.

```
even veni enif nifn ifno fnon none
     what hati atis tisi isit
```

Kemudian gunakan hasil ini sebagai masukan himpunan string dari algoritma RabinKarpSet seperti yang sudah dijelaskan pada bab sebelumnya. Diperoleh masukan algoritma RabinKarpSet-nya seperti ini:

```
 teks: evenifnoneofthesethings...computer
set s: {even, veni, enif, nifn, ...,uter}
                    m: 4
```

Selanjutnya, sesuai dengan algoritma, lakukan *hashing* ke seluruh pecahan string pada *set* s. Fungsi *hash* yang diberikan inilah yang merupakan kunci dalam menemukan pola kalimat pada teks, sehingga pastikan agar fungsi *hash* memadai untuk setiap string pada *k-gram* yang dihasilkan. Berikut contoh *hashing* untuk string-string pada *set* s (agar memudahkan, nilai *hash* string yang diberikan acak dan berharga kecil).

```
12 22 13 15 35 26 18
```

Dengan cara pengulangan iteratif sampai mencapai akhir teks, nilai *hash* string-string pada *set* s dicocokkan dengan nilai *hash* penggalan string sepanjang 4 karakter pada teks (benar apabila nilai *hash* penggalan teks asli terdapat pada himpunan nilai *hash* string-string masukan). Kita sebut saja hs bagi himpunan nilai *hash* untuk *set* s.

```
hash(even) ∈ hs    hash(veni) ∈ hs
hash(enif) ∈ hs    hash(nifn) ∈ hs
hash(ifno) ∈ hs    hash(fnon) ∈ hs
                ...
hash(youl) !∈ hs   (bukan elemen hs)
hash(ould) !∈ hs   (bukan elemen hs)
hash(uldb) !∈ hs   (bukan elemen hs)
hash(ldbe) !∈ hs   (bukan elemen hs)
hash(dbes) !∈ hs   (bukan elemen hs)
                ...
hash(pute) ∈ hs    hash(uter) ∈ hs
```

Pada setiap penggalan teks asli sepanjang 4 karakter, dilakukan *hashing* dengan fungsi *hash* yang sama seperti yang dilakukan terhadap string-string *set* s. Pencarian nilai *hash* penggalan teks asli pada hs dapat dilakukan dengan algoritma pencarian biasa, sesuai kehendak. Hasil akhir yang didapat untuk perbandingan dokumen pada Tabel 1 adalah:

Tabel 2 Hasil perbandingan dokumen mahasiswa dan dokumen asli dari *website*

Dokumen mahasiswa
Even if none of *these things happen to you,* youd *be surprised at what you might find lurking on your computer!*
What are *malware*, adware, and spyware? *Malware* is a *short for malicious software* and is a catch all phrase to describe *any software* that is designed to damage a *computer*.
Dokumen asli dari http://www.jellico.com/spyware.html
Even if none of *these things happen to you,* you'ld *be surprised at what you might find lurking on your computer!*
What is it? *Malware* (*short for* "*malicious software*") is *any software* developed for the purpose of doing harm to a *computer*.

Dari hasil perbandingan kedua potongan dokumen, terlihat sangat jelas bahwa mahasiswa mengopi sebagian besar dokumen asli. Pola tulisan mahasiswa menunjukkan kemiripan dengan pola penulisan pada dokumen asli, sehingga dapat disimpulkan ia telah melakukan tindakan plagiat.

4. PENINGKATAN AKURASI LANJUTAN

Untuk peningkatan akurasi pencarian pola dokumen plagiat, dapat dikembangkan varian dari algoritma Rabin-Karp untuk menyesuaikan dengan jenis dokumen. Salah satu contohnya menyesuaikan *filtering* dengan menambah atau mengubah kata-kata apa saja yang dapat diabaikan dan yang tidak. Selain itu dapat pula melakukan teknik pengelompokkan nilai *hash*. Teknik ini digunakan untuk mengelompokkan nilai *hash* string sesuai keterurutan pada teks asli sehingga perbandingan yang dilakukan adalah perbandingan per kelompok dengan ketentuan tertentu seperti, kelompok 0 *mod* 5 (hasil *modulo* dengan 5 = 0) dan sebagainya, tentunya disesuaikan dengan fungsi *hash*.

5. KESIMPULAN

Kemiripan pola antar dua buah dokumen dapat dicari dengan menerapkan prinsip algoritma pencarian string Rabin-Karp. Algoritma Rabin-Karp menghasilkan efisiensi waktu yang baik dalam mendeteksi string yang memiliki lebih dari satu pola. Hal ini membuat algoritma Rabin-Karp dimanfaatkan dalam melakukan pendeteksian terhadap tindak plagiat dokumen.

REFERENSI

[1] Noynaert, J.Evan, “Plagiarsim Detection Software”, http://www.cs.uwec.edu/mics/CD%20image/papers/paper97.pdf, waktu akses: 16 Mei 2008 pukul 15.00.

[2] http://www.sparknotes.com/cs/searching/hashtables/, waktu akses: 16 Mei 2008 pukul 16.00.

[3] Aiken, Alex, Saul Scheimer, dan Daniel S. Wilkerson, “Winnowing: Local Algorithms for Document Fingerprinting”, http://www.theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf, waktu akses: 16 Mei 2008 pukul 15.00.

[4] Munir, Rinaldi, “Diktat Kuliah IF2251 Strategi Algoritmik”, Departemen Teknik Informatika ITB, 2007.

[5] http://en.wikipedia.org/wiki/, waktu akses: 16 Mei 2008 pukul 14.00.