BAB II

LANDASAN TEORI

2.1 Plagiarisme

Menurut Sulianta (2007), kata plagiarisme berasal dari kata Latin *plagiarius* yang berarti merampok, membajak. Sedangkan menurut Sastroasmoro (2007), plagiarisme adalah bentuk penyalahgunaan hak kekayaan intelektual milik orang lain, yang mana dipresentasikan dan diakui secara tidak sah sebagai hasil karya pribadi. Orang yang melakukan plagiarisme disebut plagiaris atau plagiator.

Menurut Dian Novian (2012), terdapat beberapa jenis plagiarisme yang dikenal selama ini, yaitu sebagai berikut.

a. *Word-for-word plagiarism* : menyalin setiap kata secara langsung tanpa diubah sedikitpun
b. *Plagiarism of the form of a source* : menyalin dan atau menulis ulang kode-kode program tanpa mengubah struktur dan jalannya program
c. *Plagiarism of authorship*: mengakui hasil karya orang lain sebagai hasil karya sendiri dengan cara mencantumkan nama sendiri menggantikan nama pengarang sebenarnya.

Dalam hal pemrograman, terdapat 2 jenis teknik plagiat yang umum dilakukan, yaitu sebagai berikut.

1. Leksikal

Teknik plagiat leksikal merupakan teknik plagiat dengan melakukan pengubahan kode program tanpa mengubah strukturnya. Beberapa contoh tindakan plagiarisme leksikal yaitu dengan mengubah penamaan variabel,

menambahkan atau mengurangi komentar, mengubah cara deklarasi variabel, dan lainnya.

2. Struktural

Teknik plagiat struktural merupakan teknik plagiat dengan mengubah struktur kode program tanpa mengubah fungsi program. Beberapa contoh plagiat struktural seperti mengubah susunan metode, memindahkan isi kode prosedur atau fungsi ke dalam kode utama dan sebaliknya.

2.2 Algoritma Hamming Distance

Algoritma *hamming distance* merupakan salah satu dari algoritma *approximate string matching* yang ditemukan oleh Richar Hamming, pada tahun 1950. Algoritma *hamming distance* pertama kali digunakan untuk mendeteksi dan memperbaiki telekomunikasi sebagai estimasi *error* Primadani (2014).

Algoritma *hamming distance* dapat digunakan untuk menghitung jumlah perbedaan dua buah biner yang mempunyai panjang yang sama James (2013). Menurut D.H. Murti (2005), rumus yang digunakan pada algoritma ini dapat dilihat pada Rumus 2.1.

$$d_{ij} = q + r \quad \text{...(2.1)}$$

d_{ij}= nilai *hamming distance*

q = Jumlah variabel dengan nilai 1 pada objek ke-i tapi bernilai 0 pada objek ke-j.

r = Jumlah variabel dengan nilai 0 pada objek ke-i tapi bernilai 1 pada objek ke-j.

Menurut penelitian Pandiselvam.P (2014), algoritma *hamming distance* memiliki kompleksitas $O(N^2)$ komplesitas ini terbilang kompleks bahkan terbilang

paling kompleks diantara algoritma *string matching* lainnya. Namun demikian, algoritma *hamming* distance memiliki tingkat akurasi yang paling baik dari beberapa algoritma *string matching* lainnya. Tabel 2.1 menunjukkan perbandingan beberapa algoritma dalam melakukan *string matching*.

Tabel 2.1 Perbandingan Kinerja Algoritma *String Matching*

Algorithm	*Pre-processing*	*Execution Time*	*Accuracy*
Hamming	None	$O(N^2)$	81%
Levenshtein	None	O(N+M)	70%
Needleman wunsch	None	O(MN)	60%
Smith waterman	None	O(MN)	71.4%
Knuth Morris Pratt	O(M)	O(M+N)	65%
Brute Force	None	O(MN)	66.7%
Boyer Moore	O(M+N)	O(MN)	75%
Rabin Karp	O(N)	O(MN)	70%
Aho-Corasick	None	O(N+M +Z)	61.8%
CommentZ Walter	None	O(N+M+Z)+ O(MN)	61.8%

(Sumber: A Comparative Study on String Matching Algorithms of Biological Sequences)

2.3 Algoritma Brute Force

Algoritma *brute force* menurut Abdeen (2011) adalah algoritma paling sederhana yang dapat digunakan pada pencarian pola. Meskipun algoritma *brute force* merupakan algoritma yang dapat diimplementasikan pada hampir semua permasalahan, namun algoritma ini sebenarnya bukanlah algoritma yang cerdas dan efisien, sebab tidak jarang dalam prosesnya algoritma ini memakan banyak sumber daya untuk menyelesaikan masalah.

Menurut Sinapova (2014) salah satu algoritma yang termasuk ke dalam *brute force* adalah algoritma *sequential search.* Algoritma ini bekerja dengan cara membandingkan elemen-elemen dalam *list* dengan sebuah kata kunci sampai menemukan kata kunci yang dicari atau tidak ditemukan sama sekali. Adapun kompleksitas terburuk dari algoritma ini adalah O(n), sedangkan kompleksitas terbaik dari algoritma ini adalah O(1) bergantung kepada lokasi elemen yang dicari.

2.4 Preprocessing

Sebagian besar teknik deteksi kemiripan kode program saat ini melakukan *preprocessing* untuk membantu meningkatkan akurasi yang lebih baik Dani (2006)**.** Penggunaan teknik *preprocessing* ini memang akan menambah waktu proses sistem, namun teknik *preprocessing* ini cukup ampuh dalam mendeteksi plagiarisme yang bersifat leksikal terhadap suatu *source code.* Beberapa contoh plagiarisme leksikal antara lain perubahan nama variabel, penambahan atau penghapusan komentar, mengubah logika *looping for* menjadi *while* atau sebaliknya, dan lainnya, sehingga untuk mendeteksi plagiat seperti ini dibutuhkan teknik untuk menormalisasi *souce code.* Sebab, algoritma yang digunakan dalam

mendeteksi plagiarisme hanyalah algoritma pencocokan *string*, dimana algoritma tersebut membandingkan *string* secara eksplisit.

Sedangkan dalam penelitain Arifianto (2011), disebutkan bahwa dalam plagiarisme bahasa pemrograman itu sendiri, tidak jarang dilakukan modifikasi-modifikasi kecil untuk menghindari deteksi. Untuk itulah, dibutuhkan algoritma buatan untuk mendukung dan mengoptimalkan deteksi plagiarisme. Implementasi dari teknik *preprocessing* ini dapat bervariasi bergantung kepada kode program yang akan dideteksi.