Pencarian Kata yang Paling Sering Muncul dengan Algoritma Boyer-Moore dan Algoritma Knuth-Morris-Pratt

Fikri Aulia (13513050)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13513050@std.stei.itb.ac.id

***Abstrak*—*Searching*/Pencarian merupakan sebuah permasalahan yang umum terjadi terutama dalam dunia computer. Terdapat banyak algoritma untuk menyelesaikan permasalahan ini dan tentunya dengan kompleksitas yang berbeda-beda. Termasuk juga permasalahan dalam mencari kata dalam sebuah kumpulan kata atau kalimat.**

***Kata Kunci*—Searching, kompleksitas, *Binary Search, Merge Search*.**

I. PENDAHULUAN

menentukan kata apa yang paling banyak terdapat pada suatu kalimat atau kumpulan data sangat membantu dalam banyak hal. salah satunya untuk menentukan hot topic dari suatu artikel. Tidak hanya itu, dengan adanya algoritma ini juga akan sangat membantu baik pemerintah atau perusahaan dalam proses survey apa pembicaraan yang sedang hangat-hangatnya di media sosial seperti facebook atau twitter.

Berbeda dengan spesifikasi Tugas Besar III IF2211 tahun 2015. Pada makalah ini menjelaskan bagaimana kita dapat memperoleh hot-topik tanpa harus memberikan kata kunci. Hal ini akan sangat mempermudah dalam aplikasi di dunia nyata jika kita ingin mendapatkan hot topic secara keseluruhan tanpa harus memasukkan kata kunci terlebih dahulu.

II. TEORI

2.1. Algoritma Pencarian String

Algoritma pencarian *string* adalah salah satu jenis algoritma yang mencari sebuah pola dalam suatu pola yang lebih besar. Terdapat berbagai jenis algoritma yang dapat digunakan seperti *brute force, Knuth-Morris-Pratt, Boyer-Moore, Rabin-Karp, AhoCorasick,* dan lain-lain. Penerapan algoritma pencarian pola ini dapat ditemukan dalam berbagai bidang seperti biologi dalam pencarian untaian DNA, informatika dalam *search engine*, dan pendeteksian plagiarism dalam karya tulis.

2.2 Algoritma Knuth-Morris-Pratt

Algoritma *Knuth-Morris-Pratt* adalah algoritma pencarian *string* yang mencari dengan cara menghitung dari dimulai dari ketidakcocokan ditemukan, dari ketidakcoockan tersebut akan dhitung dari mana pencarian selanjutnya sebaiknya dimulai. Algoritma *Knuth-Morris-Pratt* menggunakan fungsi pembatas (*border function)* yang digunakan untuk menghitung urutan keberapa perbandingan harus dilakukan. *Border function* dihitung dengan menghitung *panjang prefix* yang ada disebuah *pattern* yang sama dengan *suffix*-nya.

Contoh Algoritma KMP dalam Java:

```
public static int kmpMatch(String text, String pattern)
  {   int n = text.length();   int
m = pattern.length();   int
fail[] = computeFail(pattern);
   int i=0;
int j=0; while
(i < n) {
    if (pattern.charAt(j) ==
text.charAt(i)) {       if (j == m - 1)
return i - m + 1; // match       i++;
j++;    }     else if (j > 0)      j =
fail[j-1];     else      i++;
   }
   return -1; // no match
 } public static int[] computeFail(String
   pattern)
 {
   int fail[] = new int[pattern.length()];
   fail[0] = 0;
   int m =
pattern.length();   int
j = 0;   int i = 1;
      while (i < m) {
    if  (pattern.charAt(j) ==
pattern.charAt(i)) {   //j+1 chars match
      fail[i] = j
+ 1;       i++;
j++;
    }
    else if (j > 0) // j follows matching prefix
      j = fail[j-1];
else {   // no
match       fail[i] =
0;       i++;     }
}   return fail;  }
```

2.3 Algoritma Boyer-Moore

Algoritma *Boyer-Moore* adalah algoritma pencarian *string* yang mencari dengan cara membandingkan sebuah huruf dengan huruf yang ada di *pattern* yang dicari, dan menggeser *pattern* tersebut hingga posisinya sama dengan teks yang dicari dan membandingkan kata tersebut. Cara ini disebut *character jump.*

Contoh Algoritma *Boyer-Moore* dalam *Java:*

```
public static int bmMatch(String text, String pattern)
  {int last[] = buildLast(pattern);
int n = text.length();     int m =
pattern.length();     int i = m-1;     if (i
> n-1)       return -1;     int j = m-1;
do {       if (pattern.charAt(j) ==
text.charAt(i))        if (j == 0)
return i; // match        else {          i--;
j--;
       }
     else { int lo =
last[text.charAt(i)];          i = i + m -
Math.min(j, 1+lo);
        j = m - 1;
     }
   } while (i <= n-1);     return -1;    }
public static int[] buildLast(String
pattern)
 {     int last[] = new int[128]; // ASCII
char set


   for(int i=0; i < 128; i++)
last[i] = -1; // initialize array


   for (int i = 0; i < pattern.length(); i++)
last[pattern.charAt(i)] = i;


   return last;
  }
```

III. ALGORITMA PENCARIAN

Algoritma pencarian menggunakan Boyer-Moore dan Knuth-Morris-Praat yang dimodifikasi. Perubahan /modifikasi terdapat pada output dari program algoritma yang ada dan mengupayakan agar algoritma tetap berjalan walaupun pattern atau sub kata yang dicari sudah ditemukan.

Berikut Algoritma yang merupakan hasil modifikasi tersebut

Algoritma Boyer-Moore

```
public static int bmMatch(String text, String pattern){
int last[] = buildLast(pattern);
int n = text.length();
int m = pattern.length();
int i = m-1;
int count = 0;
        if (i > n-1){
                return count;
        }
        int j = m-1;
        do {
                //if (i<0) i = 0;
                //if (j<0) j = 0;
                if (pattern.charAt(j) == text.charAt(i)){
                        if (j == 0){
                            count++;
                            i+=pattern.length();
                        } else {
                            i--;
                            j--;
                        }
                  } else {
                        int lo = last[text.charAt(i)];
                        i = i+m-Math.min(j, 1+lo);
                        j = m-1;
                  }
            } while (i <= n-1);
            return count;
        }

        public static int[] buildLast(String pattern){
                int last[] = new int[128];
                for(int i=0; i<128; i++){
                        last[i] = -1;
                }
                for(int i=0; i< pattern.length(); i++){
                        last[pattern.charAt(i)] = i;
                }
                return last;
        }
```

Algoritma Knutch-Morris-Praat

```
public static int kmpMatch(String text, String pattern){
    int n = text.length();
    int m = pattern.length();
    int fail[] = computeFail(pattern);
    int i = 0;
    int j = 0;
    int count = 0;
    while (i < n){
            if (pattern.charAt(j) == text.charAt(i)){
                    if (j == m - 1){
                            count++;
                            i++;
                            j = 0;
                    }
                    i++;
                    j++;
            } else if (j > 0){
                    j = fail[j-1];
            } else {
                    i++;
            }
    }
    return count;
}

public static int[] computeFail(String pattern){
    int fail[] = new int[pattern.length()];
    fail[0] = 0;
    int m = pattern.length();
    int j = 0;
    int i = 1;
    while (i < 0){
            if (pattern.charAt(j) == pattern.charAt(i)){
                    fail[i] = j +1;
                    i++;
                    j++;
            } else if (j > 0) {
                    j = fail[j-i];
            } else {
                    fail[i] = 0;
                    i++;
            }
    }
    return fail;
}
```

Proses pencarian dimulai dengan mengenerate kalimat atau string menjadi list of string, sehingga dengan adanya list of string maka kita dapat melakukan pengecekan terhadap tiap-tiap string tehadap kalimat yang akan dicek. Perlu menjadi perhatian bahwa isi list of string adalah unik.

Sehingga out putnya nanti adalah list of string beserta banyaknya jumlah kemunculan kata tersebut. Memungkinkan juga kita untuk melakukan proses sorting untuk mengurutkan kata apa yang paling banyak muncul mengecil atau membesar. Berikut algoritma pencarian yang sudah lengkap :

```
import java.util.Scanner;
import java.util.Arrays;
import java.lang.Object;
import org.apache.commons.lang3.ArrayUtils;
public class MakalahStima{
public static int kmpMatch(String text, String
pattern){
int n = text.length();
int m = pattern.length();
int fail[] = computeFail(pattern);
int i = 0;
int j = 0;
int count = 0;
while (i < n){
        if (pattern.charAt(j) == text.charAt(i)){
                if (j == m - 1){
                        count++;
                        i++;
                        j = 0;
                }
                i++;
                j++;
        } else if (j > 0){
                j = fail[j-1];
        } else {
                i++;
        }
}
return count;
}
public static int[] computeFail(String pattern){
int fail[] = new int[pattern.length()];
fail[0] = 0;
int m = pattern.length();
int j = 0;
int i = 1;
while (i < 0){
        if (pattern.charAt(j) == attern.charAt(i)){
                fail[i] = j +1;
                i++;
                j++;
        } else if (j > 0) {
                j = fail[j-i];
        } else {
                fail[i] = 0;
                i++;
        }
}
return fail;
}
public static int bmMatch(String text, String
pattern){
int last[] = buildLast(pattern);
int n = text.length();
int m = pattern.length();
int i = m-1;
int count = 0;
if (i > n-1){
        return count;
}
int j = m-1;
do {
        //if (i<0) i = 0;
        //if (j<0) j = 0;
        if (pattern.charAt(j) == text.charAt(i)){
                if (j == 0){
                        count++;
                        i+=pattern.length();
                } else {
                        i--;
                        j--;
                }
        } else {
                int lo = last[text.charAt(i)];
                i = i+m-Math.min(j, 1+lo);
                j = m-1;
        }
} while (i <= n-1);
return count;
}

public static int[] buildLast(String pattern){
        int last[] = new int[128];
        for(int i=0; i<128; i++){
                last[i] = -1;
        }
for(int i=0; i< pattern.length(); i++){
        last[pattern.charAt(i)] = i;
}
        return last;
}
public String[] GenerateText(String text){
String[] generated = new String[100];
String[] temp = new String[1];
int i = 0;
String subText ="";
```

```
while (i <= text.length()){
        if (text.charAt(i) != ' '){
                subText = subText + text.charAt(i);
        } else {
                temp[0] = subText;
                generated =
ArrayUtils.addAll(generated, temp);
                subText = "";
                }
        }
}

public static void main(String[] args) {
Scanner scan = new Scanner(System.in);
int count[] = new int[100];
String text = scan.next();
String[] generatedText = GenerateText(text);
int pos = 0;
while (generatedText[pos] != null){
        count[pos] = bmMatch(text,
generatedText[pos]);
}
for(int i = 0; i< pos; i++){
        System.out.println("kata : " +
generatedText[pos] + "dengan jumlah : " +
count[pos]);
        }
        }
}
```

IV. KEKURANGAN

Pada Agoritma ini masih terdapat pemborosan, hal ini terjadi karena mengharuskan adanya pengecekan kata sebanyak 2 kali. Ini terjadi karena proses pencarian kata yang akan dimasukkan kedalam array of string. Sehingga ini mengakibatkan pencarian menjadi 2 kali lipat lebih lama. Hal ini akan terasa akibatnya jika string yang menjadi masukkan adalah string yang sangat panjang.

Kekurangan lain adalah belum adanya implementasi langsung terhadap target penggunaan algoritma ini sendiri. Proses pengecekan masih terbatas input dari penulis. Sehingga kemungkinan-kemungkinan kesalahan karena factor luar belum teratasi. Sehingga algoritma ini masih rentan akan kesalahan.

V. KESIMPULAN

Dari sekian banyak algoritma yang kita pelajari, tidak menutup kemungkinan untuk dilakukannya modifikasi agar mampu menyelesaikan permasalahan yang kita hadapi. Namun harus tetap memegang dasar dari algoritma yang kita gunakan tersebut. Dari makalah ini dapat kita lihat bagaimana algoritma KMP dan BM kita ubah sedemikian rupa sehingga dapat digunakan untuk menyelesaikan masalah yang ada.

VII. UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Tuhan YME karena atas isin-Nya makalah ini dapat selesai.. Penulis mengucapkakn terima kasih kepada bapak Rinaldi Munis atas bimbingannya selaku dosen Stategi ALGORITMA serta kepada pendahulu-pendahlu yang telah memberikan karya-karya terkait algoritma penyelesaikan berbagai masalah terutama dalam masalah pencarian sehingga membantu dalam menyelesaikan makalah ini.

DAFTAR PUSTAKA

[1] Munir, Rinaldi. *Diktat Kuliah IF2091 Strategi Algoritma*. Bandung : Penerbit informatika. 2008

[2] Happy Atrinawati, Lovita. *Analisis Kompleksitas Algoritma untuk Berbagai Macam Metode Pencarian Nilai(Searching) dan Pengurutan Nilai (Sorting) pada Tabel*. Bandung : ITB , 2005

[3] "[pdf] Algoritma Pencocokan String (String Matching Algorithm) - Lecturer EEPIS" ~ lecturer.eepis-

[4] www.softpanorama.org

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 4 Mei 2015

Fikri Aulia 13513050