Penerapan Algoritma Pattern Matching untuk Identifikasi Lagu

Wiega Sonora – 13514019
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
wiegasonora@gmail.com

***Abstract*—Di dalam dunia pemrograman terdapat istilah algoritma. Algoritma merupakan cara – cara yang digunakan untuk menyelesaikan permasalahan tertentu dan disusun secara terstruktur dan efektif. Salah satu jenis algoritma yang cukup terkenal dan banyak digunakan untuk menyelesaikan beragam permasalahan ialah algoritma pattern matching. Algoritma pattern matching digunakan untuk melihat apakan ada kecocokan antara string pattern pada string text. Salah satu penggunaan dari algoritma ini ialah mencocokan string biner dari penggalan lagu yang didapat dari rekaman untuk mengidentifikasi lagu tersebut dengan mencocokannya ke string biner daftar lagu secara keseluruhan.**

Keywords— Algoritma, Pattern Matching, KMP, Boyer Moore, Identitas Lagu, Shazam.

I. PENDAHULUAN

Seringkali pada saat kita bersantai kita senang mendengarkan lagu yang diputar melalui radio atau televisi. Terkadang pada saat mendengarkan lagu tertentu, kita tertarik untuk mengetahui identitas lagu tersebut mulai dari penyanyi, judul lagu, album, serta informasi penting lainnya. Sayangnya jika kita mendengarkan melalui radio atau televisi seringkali tidak ada informasi mengenai judul lagu atau penyanyi yang membawakan lagu tersebut. Hal ini menjadi masalah apabila kita ingin mendengarkan atau mencari lagu tersebut di waktu yang akan dating.

Perkembangan mobile apps di jaman sekarang sangat pesat hingga memungkinkan untuk membuat beragam jenis aplikasi untuk kebutuhan – kebutuhan tertentu. Salah satu jenis mobile apps yang sering diunduh dan digunakan adalah pencocokan lagu. Jenis mobile apps seperti ini digunakan untuk merekam lagu yang diputar di sekitar kita lalu mencocokannya dengan katalog lagu pada server kemudian menghasilkan keluaran idntitas lagu seperti judul lagu, penyanyi, album, tahun rilis, dan lain-lain. Perkembangan mobile apps jenis ini cukup pesat dan membantu kebutuhan sebagian orang yang bermaksud untuk mengidentifikasi lagu tertentu.

Gambar 1 Tampilan Antar Muka Beberapa Aplikasi Pencocokan Lagu - *sumber: http://i.kinja-img.com/gawker-media/image/upload/s--QUV0pjVs--/18ixmi4419c29png.png*

Dengan berkembang pesatnya mobile apps yang mendukung fitur identifikasi lagu, kebutuhan untuk mengetahui identitas lagu melalui pencocokan antara rekaman penggalan lagu dengan koleksi lagu yang dimiliki server dapat ditangani. Algoritma pattern matching khususnya pada string matching lah yang membuat hal tersebut dapat dilakukan. Algoritma ini yang akan menjadi solusi dari permasalahan identifikasi lagu yang orang – orang khawatirkan.

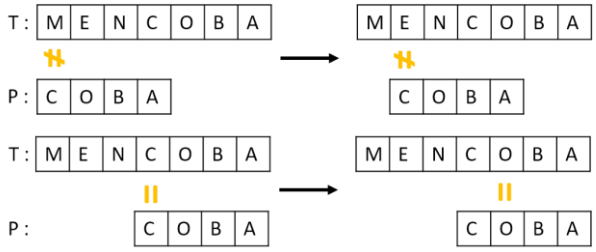
II. TEORI DASAR

A. Algoritma Pattern Matching

Jenis algoritma yang ada di dunia ini sungguh sangat banyak, antara lain : sorting, searching, branch and bound, greedy, string matching, dll. Salah satu yang akan dibahas pada makalah ini adalah algoritma string matching. Jenis algoritma ini digunakan untuk khasus yang sedang dibahas yaitu identifikasi lagu sesuai penggalan lagu yang didapat. Penggunaan algoritma string matching dapat mengecek ada atau tidaknya pola pada suatu text panjang. Terdapat beberapa variasi algoritma string matching, antara lain : Brute Force, Knuth-Morris-Pratt, serta Boyer Moore.

1. Algoritma Brute Force

Algoritma brute force seringkali disebut algoritma naif karena algoritma ini mengecek segala kemungkinan secara traversal tanpa memedulikan efisiensi dari algoritma tersebut.



Gambar 2. Alur Pencocokan String pada Algoritma Brute Force

Algoritma ini bekerja dengan alur mencocokan antara pattern dengan text huruf demi huruf mulai dari P[1] dengan T[1] hingga akhir huruf. Pencocokan dilakukan perhuruf dari awal pattern misal P[j] dengan T[i] dan pattern bergeser jika mismatch (P[j] dengan T[i+1]) sedangkan pencocokan dilakukan untuk huruf berikutnya pada pattern jika match (P[j+1] dengan T[i+1]). Untuk lebih jelasnya dapat dilihat penerapan algoritma Brute Force pada bahasa Java di bawah ini.

```
public static int brute(String text,String
pattern) {

int n = text.length();     // n is length of text
int m = pattern.length(); // m is length of
pattern
int j;
for(int i=0; i <= (n-m); i++) {
  j = 0;
  while ((j < m) && (text.charAt(i+j)==
pattern.charAt(j)) ) {
         j++;

    }
  if (j == m){
         return i;  // match at i
  }

}
   return -1;  // no match
} // end of brute()
```

Gambar 3 Algoritma Brute Force String Matching pada Bahasa Java - *disadur dari Slide Kuliah Pattern Matching oleh Dr. Andrew Davison yang dimodifikasi oleh Dr. Rinaldi Munir*

Kompleksitas waktu yang dihasilkan dalam penggunaan algoritma ini beragam tergantung dari use case yang digunakan. Untuk kasus terburuk, kompleksitas waktunya O(mn) dengan jumlah perbandingan m(n-m+1). Kasus tersebut bisa terjadi untuk pattern yang selalu mangalami match tiap kali dicocokkan ke text tetapi pada akhir pattern terdapat mismatch, oleh karena itu pattern akan mengalami pencocokan yang banyak seiring bergesernya pattern. Contoh : T: aaaaaaaaz P: aaz.

Pada percobaan dengan kasus terbaik, pencocokan antara pattern dengan text akan langsung mengalami mismatch ketika dicocokkan antara P[1] dengan T[i] sehingga jumlah perbandingan karakter bisa lebih sedikit dari biasanya. Contoh: T: Alfabet diakhiri dengan xyz. P : xyz. Kompleksitas waktu pada kasus ini adalah O(n).

Pada kasus biasa yaitu dengan pattern dan teks biasa atau normal. Kompleksitasnya secara umum adalah O(m+n). Pencocokan string dengan algoritma brute force akan cepat dilakukan untuk kasus percobaan pattern dan text alphabet sedangkan pada percobaan dengan biner (01) akan lambat.

2. Algoritma Knuth – Morris – Pratt

Algoritma string matching yang juga banyak digunakan ini ditemukan oleh Donald E. Knuth, James H. Morris and Vaughan R. Pratt pada 1977. Algoritma KMP disebut-sebut lebih efiisen dibandingkan dengan algoritma Brute Force karena pada algoritma ini tidak perlu menelusuri semua huruf pada text. Prinsip algoritma ini yaitu menentukan jumlah pergeseran pattern pada saat mencocokannya dengan text dengan terlebih dahulu menghitung fungsi pinggiran dari pattern.

Sebelum melakukan string matching, algoritma ini akan menghitung fungsi pinggiran dari masing-masing huruf pada pattern. Fungsi pinggiran berupa angka yang merepresentasikan ukuran dari prefix terbesar dari P[1..k] yang juga merupakan suffix dari P[1..k]. Alasan digunakannya fungsi pinggiran adalah untuk menghindari pengulangan pencocokan prefix (awalan) yang ada pada suffix (akhiran). Cara menghitung fungsi pinggiran digambarkan pada algoritma di bawah ini.

```
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;
 } // end of computeFail()
```

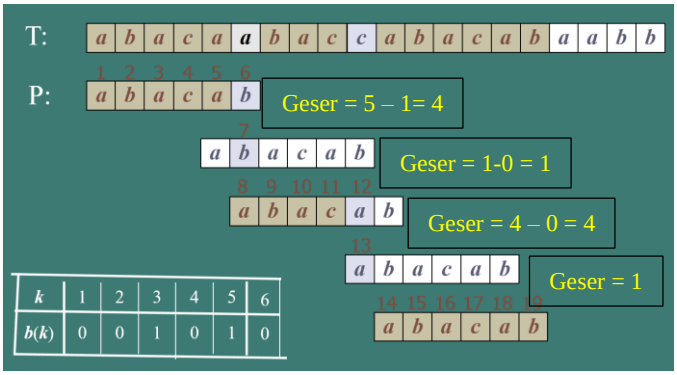
Gambar 4. Algoritma Penghitung Fungsi Pinggiran dalam Bahasa Java - *disadur dari Slide Kuliah Pattern Matching oleh Dr. Andrew Davison yang dimodifikasi oleh Dr. Rinaldi Munir*

Setelah membentuk fungsi pinggiran, algoritma ini akan mulai melakukan pencocokan. Seperti biasa, dimulai dari P[1] ketika ditemukan ketidakcocokan, maka akan dilakukan pergeseran. Jumlah pergeseran bergantung pada indeks huruf terakhir yang cocok, maka jumlah pergeseran adalah indeks dari huruf terakhir yang cocok (indeks dimulai dari 1) dikurangi dengan fungsi pinggirannya. Jika mismatch terjadi pada P[1] maka digeser 1 karakter ke kanan.

```
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
 } // end of kmpMatch()
```

Gambar 5 Algoritma Knuth-Morris-Pratt dalam Bahasa Java - *disadur dari Slide Kuliah Pattern Matching oleh Dr. Andrew Davison yang dimodifikasi oleh Dr. Rinaldi Munir*

Alur pencocokan dan pergeseran dari algoritma KMP dapat dilihat di bawah ini.



Gambar 6. Contoh Urutan Proses Pencocokan Algoritma KMP - *disadur dari Slide Kuliah Pattern Matching oleh Dr. Andrew Davison yang dimodifikasi oleh Dr. Rinaldi Munir dengan perubahan seperlunya*

Kompleksitas waktu yang dihasilkan oleh algoritma KMP ini adalah O(m+n) dengan O(m) untuk mencari fungsi pinggiran serta O(n) untuk pencarian string. Algoritma KMP tidak perlu bergerak mundur pada input teks sehingga pemrosesan algoritma ini bisa berjalan dengan cepat bahkan untuk berkas yang ukurannya besar. Sayangnya, dengan bertambahnya jenis alphabet pada teks yang akan dicocokkan, kemungkinan untuk terjadinya mismatch akan lebih tinggi sehingga algoritma ini tidak berjalan secara optimal.

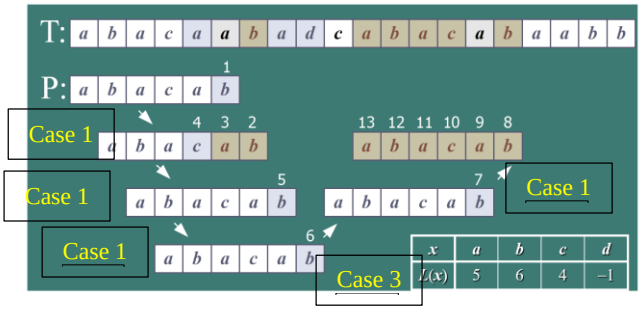
3. Algoritma Boyer – Moore

Algoritma Boyer – Moore merupakan algoritma pencocokan string selain brute force dan KMP. Algoritma ini juga dinilai cepat untuk menyelesaikan permasalahan string matching. Algoritma ini dipublikasikan oleh Robert S. Boyer, dan J. Strother Moore pada tahun 1977. Algoritma Boyer Moore terdiri dari 2 teknik, yang pertama adalah teknik looking glass yang mana berarti pencocokan patern pada text dimulai dari P[m] hingga P[1] (dari belakang). Teknik kedua adalah character-jump. Teknik ini memungkinkan pergeseran karakter yang bergantung pada tiga buah kasus.

```
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; // no match if pattern is
          // longer than text
  int j = m-1;
  do {
   if (pattern.charAt(j) == text.charAt(i))
    if (j == 0)
     return i; // match
    else { // looking-glass technique
     i--;
     j--;
    }
   else { // character jump technique
    int lo = last[text.charAt(i)]; //last occ
    i = i + m - Math.min(j, 1+lo);
    j = m - 1;
   }
  } while (i <= n-1);

  return -1; // no match
 } // end of bmMatch()
```

Gambar 7 Algoritma Boyer Moore dalam Bahasa Java - *disadur dari Slide Kuliah Pattern Matching oleh Dr. Andrew Davison yang dimodifikasi oleh Dr. Rinaldi Munir*

Pada teknik character-jump, pergeseran dilakukan menurut kasus yang terjadi. Ketiga kasus tersebut berurutan sehingga kasus yang dilakukan pertama adalah kasus 1, kemudian 2 jika kasus 1 tidak berlaku, lalu kasus 3 jika kasus 1 dan kasus 2 tidak berlaku, Kasus pertama ialah apabila Pencocokan pada P[i] dan T[i] terjadi mismatch dan karakter pada T[i] terdapat di sebelah kiri P[i], maka dilakukan pergeseran karakter sehingga keduanya sejajar (T[i] dan P[i_{baru}]) lalu dilakukan pencocokan lagi mulai dari belakang pattern. Pada kasus kedua, dilakukan pergeseran jika kasus 1 tidak berlaku (tidak ada karakter yang sesuai dengan T[i] pada sebelah kiri dari P[j] tetapi karakter pada T[i] terdapat di sebelah kanan P[j]. Jika hal ini terjadi maka dilakukan pergeseran sejumlah 1 karakter ke kanan sehingga P[j] akan sejajar T[i+1]. Kasus ketiga sekaligus terakhir berlaku jika kasus 1 dan 2 tidak berlaku (karakter pada T[i] tidak ada pada pattern) maka dilakukan pergeseran sehingga P[1] sejajar dengan T[i+1].



Gambar 8 Contoh Urutan Proses Pencocokan Algoritma Boyer Moore - *disadur dari Slide Kuliah Pattern Matching oleh Dr. Andrew Davison yang dimodifikasi oleh Dr. Rinaldi Munir dengan perubahan seperlunya*

Dalam menentukan pergeseran, sebagaimana telah disebutkan sebelumnya bahwa parameter untuk menentukan jumlah pergeseran adalah dengan melihat ke dalam pattern dan mencari di mana letak karakter T[i] berada, apakah di sebelah kiri P[j], atau di sebelah kanannya, atau bahkan tidak berada di manapun. Oleh karena itu, layaknya algoritma KMP, perlu dihitung fungsi tertentu. Fungsi ini disebut Fungsi Last Occurrence yang mana menghitung letak kemunculan terakhir (indeks) dari karakter – karakter ASCII pada pattern dan mengembalikan -1 jika karakter tidak terdapat pada pattern.

```
public static int[] buildLast(String pattern)
 /* Return array storing index of last
  occurrence of each ASCII char in 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;
 } // end of buildLast()
```

Gambar 9 Algoritma Penghitung Fungsi Pinggiran dalam Bahasa Java - *disadur dari Slide Kuliah Pattern Matching oleh Dr. Andrew Davison yang dimodifikasi oleh Dr. Rinaldi Munir*

Algoritma Boyer – Moore cepat untuk mencocokan pattern pada text dengan jenis alpabhet yang banyak dan akan lambat jika digunakan untuk pencocokan binary. Algoritma ini secara signifikan lebih cepat dalam pencocokan string kalimat biasa dibadingkan dengan algoritma brute force. Koompleksitas waktu pada kasus terburuk dari algoritma Boyer Moore adalah O(nm + A) dengan A merupakan jumlah alphabet yang muncul. Pada kasus normal, kompleksitas algoritma ini O(n + m).

B. Jenis – Jenis Aplikasi Pengidentifikasi Lagu

Aplikasi – aplikasi untuk mengidentifikasi penggalan lagu telah menjamur di toko – toko aplikasi seperti AppStore, Google Playstore, dan lain sebagainya. Aplikasi –aplikasi tersebut mencocokan penggalan lagu dan mengubahnya dalam bentuk string sehingga dapat dilakukan string matching dengan katalog lagu yang dimiliki server masing – masing aplikasi. Beberapa aplikasi identifikasi lagu tersebut antara lain:

a. Shazam

Semenjak dipasarkan pada tahun 2002 melalui layanan berbasis SMS. Aplikasi ini menjadi aplikasi utama untuk mengidentifikasi lagu. Aplikasi ini juga telah diunduh jutaan kali dari Playstore.

b. Soundhound

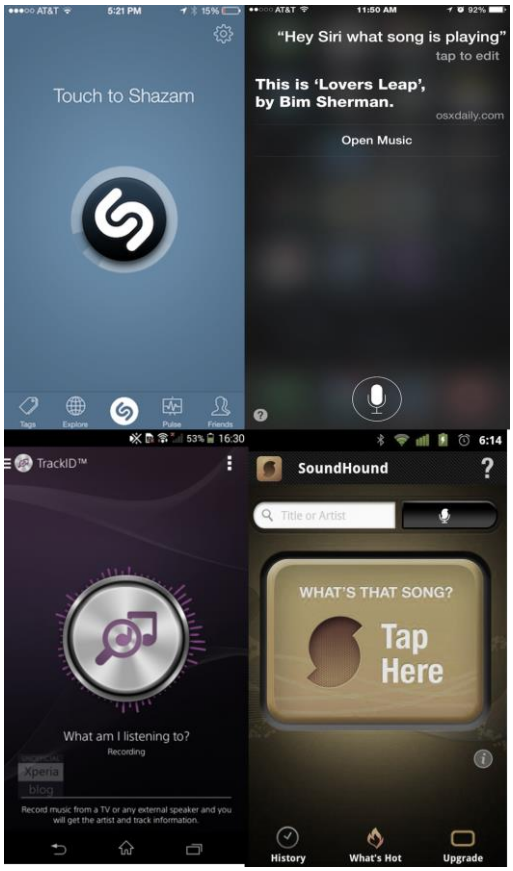
Menjadi rival Shazam, Soundhound merambah pasar aplikasi berbasis pengidentifikasi lagu.

c. Sony TrackID

Aplikasi bawaan Sony ini cukup responsive untuk digunakan mengidentifikasi lagu. Pengguna juga langsung disuguhi preview lagu berbasis Spotify.

d. Siri

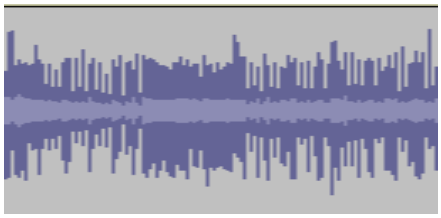
Selain berfungsi sebagai asisten pribadi, aplikasi berbasis artificial intelligence ini juga dapat digunakan untuk mengidentfikasi lagu yang sedang dimainkan.



Gambar 10. Beberapa Contoh Aplikasi Mobile untuk Identifikasi Lagu – *sumber : https://313e5987718b346aaf83-f5e825270f29a84f7881423410384342.ssl.cf1.rackcdn.com/1380403794-Shazam%207.png http://www.freedownloadapk.com/wp-content/uploads/2012/02/SoundHound-apk-for-android-2.jpg http://www.static.xperiablog.net/wp-content/uploads/2014/07/TrackID_4.0.B.1.0_7-315x560.png http://cdn.osxdaily.com/wp-content/uploads/2014/10/siri-what-song-is-playing.jpg*

III. PENERAPAN ALGORITMA PATTERN MATCHING

Sebelum melakukan pencocokan pola antara penggalan lagu yang didapat dengan cara merekam lagu dari sekitar, pola yang didapat tersebut masih berupa gelombang suara. Gelombang tersebut tidak hanya merepresentasikan lirik lagu tetapi juga nada, tempo, dan atribut laagu lainnya.



Gambar 11Contoh Gelombang Suara dari Cuplikan Lagu - *sumber: perangkat lunak Audacity*

Setelah mendapatkan gelombang suara tersebut, kemudian program akan mengubahnya ke dalam pita string yang merepresentasikan lagu tersebut (fingerprinting). String yang dimaksud berupa binary (01). Setiap aplikasi memiliki algoritma string matching sendiri dan performansinya tentu sangat efisien dikarenakan identifikasi lagu dapat dilakukan hanya dalam hitungan detik (sudah termasuk menampilkan identitas lagu yang didapat dari server). Biasanya algrotima yang digunakan adalah KMP atau Boyer-Moore yang oleh perusahaan peembuat aplikasi diberikan modifikasi untuk menambah kemangkusan algoritma tersebut.

String of Music	String of Pattern
00011101010101000011010101110 01110001101010011100011100001 10101010011000111000011010111 11010001011001010001110**00111** 10000011100001110000011110000 11110000011010101010001100 10 11001011100011100011100011000 01100001110001110001100010 01 01000111100110101000110101 01 0110101010100...	11001010001110

Gambar 12 Contoh Gambaran Pencocokan String pada Lagu

Ada kalanya rekaman dari cuplikan lagu yang didapat tidak sejernih lagu aslinya. Hal ini sangat mungkin jika pada proses merekam cuplkan lagu dari radio / televisi terdapat noise yang cukup banyak dan menggangu. Beberapa aplikasi identifikasi lagu dapat mengatasinya dengan menggunakan perhitungan

kemiripan. Dengan hal ini, algoritma string matching yang digunakan dimodifikasi sehingga tidak harus sama persis seperti patern tetapi dapat memberikan toleransi kesalahan sehingga jika pattern dengan ukuran 14 karakter memiliki kesamaan 12 karakter dengan lagu A, 10 karakter dengan lagu B, dan 6 karakter dengan lagu C, aplikasi akan menyimpulkan bahwa cuplikan lagu tersebut merupakan cuplikan dari lagu A dikarenakan jumlah kecocokannya paling besar walaupun tidak 100% mirip. Namun, apabila noise yang terkandung dalam rekaman penggalan lagu tersebut cukup besar, ada kalanya aplikasi dengan algoritmanya tidak dapat mentolerir sehingga akan megeluarkan pesan kesalahan dan pengguna harus mencoba lagi dan mengurangi gangguan suara.

Setelah menelusuri database lagu yang terdapat pada server dengan algoritma terntentu dan menemukan kecocokan tertinggi yang disinyalir merupakan lagu aslinya, aplikasi akan menampilkan informasi mengenai judul lagu, nama penyanyi / band, lengkap dengan album dan cover albumnya. Selain itu, beberapa aplikasi juga akan melakukan pencarian melalui youtube dan beberapa penyedia music berbasis online seperti Spotify. Sehingga selain identitas lagu, aplikasi juga menyediakan hyperlink menuju video music tersebut di YouTube atau dapat langsung memutar lagu itu melalui Spotify.

Penerapan pada program aslinya, tidak sesimple yang dijelaskan di atas karena algoritma string matching akan mencocokkan beberapa string untuk tiap tiap alat music yang dimainkan (gitar, piano, bass, dan lain-lain) sehingga string matching antara pattern dengan text yang didapat dari lagu aslinya seringkali dilakukan lebih dari satu kali untuk mendapatkan hasil yang pasti.

IV. KESIMPULAN

Algoritma pattern matching merupakan algoritma yang cukup banyak digunakan untuk menyelesaikan berbagai permasalahan, salah satunya adalah identifikasi lagu. Beberapa aplikasi identifikasi lagu seperti Shazam, Soundhound, dan lain-lain memanfaatkan algoritma ini untuk mencocokan cuplikan lagu yang didengar oleh pengguna dengan database lagu yang dimiliki server sehingga pengguna dapat menikmati lagu tersebut di waktu yang akan datang.

V. UCAPAN TERIMA KASIH

Pertama – tama saya mengucapkan terima kasih kepada Tuhan Yang Maha Esa yang telah melimpahkan rahmat dan hidayah-Nya sehingga makalah Strategi Algoritma ini dapat diselesaikan tepat waktu. Saya juga mengucapkan terima kasih kepada kedua orang tua saya yang selalu memberi dukungan dan doa restu kepada saya sehingga saya dapat menempuh pendidikan sampai saat ini. Tak lupa juga saya mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, MT serta Ibu Dr. Nur Ulfa Maulidevi yang berperan sebagai dosen matakuliah IF2211 Strategi Algoritma sehingga dengan ilmu pengetahuan seputar Strategi Algoritma saya dapat membuat dan menyelesaikan makalah ini.

REFERENSI

[1] *Davison Andrew*, Pattern Matching, 2006 (updated by Rinaldi Munir)

[2] *Mandumula, Kranthi Kumar.* Knuth Morris Pratt Algorithm. 2011.

[3] *Cliffor, Raphel and Iliopoulos, Costas*. Approximate String Matching for Music Analysis.

[4] *Kilpelainen, Pekka.*Lecture 3 : Boyer Moore Algorithm. 2005. University of Kuopio

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, 9 Mei 2016

Wiega Sonora - 13514019