Analisis Plagiarisme dalam Dua Buah Lagu Yang Berbeda dengan Algoritma *Pattern Matching*

Ega Rifqi Saputra (13515015)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515015@itb.ac.id

Abstrak*—Plagiarisme adalah kegiatan meniru karya orang lain dan mengakuinya sebagai karya milik dia sendiri. Plagiarisme merupakan salah satu tindakan kriminal dan hal hal tentang plagiarisme ini pun telah dituliskan didalam Undang Undang tentang Hak Cipta. Namun plagiarisme masih marak dilakukan, salah satunya adalah plagiarisme dalam lagu. Plagiarisme ini tentu merugikan sebagian orang, khususnya yang di plagiat. Akan tetapi para plagiator terkadang menepis kenyataan bahwa mereka melakukan plagiarisme terhadap karya orang lain dan mengatakan bahwa terinspirasi. Dikatakan di dalam Undang Undang, lagu yang diakatakan plagiat adalah jika memiliki kesamaan berupa 8 bar. Dengan ini, plagiarisme bisa dicek. Salah satunya adalah dengan menggunakan pattern matching. Samakan kedua buah lagu, jika mencapai angka yang cukup, maka salah satunya adalah plagiat dari yang lainnya. Pengecekkan plagiarisme dengan menggunakan pattern matching bisa menggunakan 3 macam algoritma yaitu, *brute force, Knuth – Morris – Pratt dan Boyer Moore

Kata kunci*—Plagiarisme, *Pattern, Searching, Brute Force, KMP, Boyer Moore

I. PENDAHULUAN

Plagiarisme adalah penjiplakan atau pengambilan karangan, pendapat, dan sebagainya dari orang lain dan menjadikannya seolah karangan dan pendapat sendiri. Plagiat dapat dianggap sebagai tindak pidana karena mencuri hak cipta orang lain. Di dunia pendidikan, pelaku plagiarisme dapat mendapat hukuman berat seperti dikeluarkan dari sekolah/universitas. Pelaku plagiat disebut sebagai plagiator. Yang digolongkan sebagai plagiarisme adalah menggunakan tulisan orang lain secara mentah, tanpa memberikan tanda jelas (misalnya dengan menggunakan tanda kutip atau blok alinea yang berbeda) bahwa teks tersebut diambil persis dari tulisan lain dan/atau mengambil gagasan orang lain tanpa memberikan anotasi yang cukup tentang sumbernya.

Plagiarisme ini sering kali ditemukan bahkan dalam kegiatan sehari hari. Hal hal kecil seperti menyontek saat ujian, menyontek tugas teman, mengutip tulisan teman tanpa ada penyebutan sumber sumber dan hal lainnya. Namun plagiarisme yang kebanyakan ramai dibicarakan adalah plagiarisme lagu atau musik. Plagiarism dalam musik seperti yang tercantum dalam laman Wikipedia adalah *"*penggunaan atau proses imitasi yang dilakukan pencipta lagu lain dan menyajikannya kembali sekaligus mengklaimnya sebagai karya asli sendiri*".* Plagiat sebuah karya musik memang sudah terjadi sejak dahulu, mulai dari lagu kebangsaan maupun lagu-lagiu daerah yang dianggap sebagai karya plagiat.

Undang-Undang Hak Cipta Tahun 2014 sebagai perbaikan dari Undang-Undang Hak Cipta Tahun 2002 mulai disosialisasikan. Salah satu hal yang paling ditekankan dalam sosialisasi tersebut adalah hal baru yang disebut substantial part. "Yaitu, bagian terpenting dalam musik yang pernah dikenal orang." . Sebelumnya, sebuah lagu dikatakan plagiat alias menyontek jika memiliki kesamaan dengan lagu lainnya sebanyak 8 bar. Tetapi dengan undang – undang yang baru diperbaiki ini, walaupun belum sampai satu bar, jika sudah terdengar sama dengan lagu yang lain, bisa saja dikatakan plagiat.)

Dengan berkembangnya teknologi, bisa saja dibuat suatu aplikasi yang akan mencocokkan dua buah lagu, apakah kedua lagu tersebut sama dan bisa disebut plagiat atau kedua lagu tersebut benar berbeda. Untuk melakukannya bisa dengan menggunakan pattern matching yang sudah dipelajari di kuliah strategi algoritma yaitu brute force, KMP dan Boyer Moore. Caranya adalah dengan mengambil 8 bar yang ada di salah satu bar dan mencari apakah ada yang memiliki not yang sama dengan 8 bar yang diambil. 8 bar tersbut diambil dari awal atau dilakukan secara heuristik yaitu ditentukan dari *user*. Bagian yang ingin dicek pun diiterasi dari awal atau dilakukan secara heuristik yaitu ditentukan dari *user* akan dicek dari *range* waktu sekian menit sampai sekian menit.

II. DASAR – DASAR TEORI

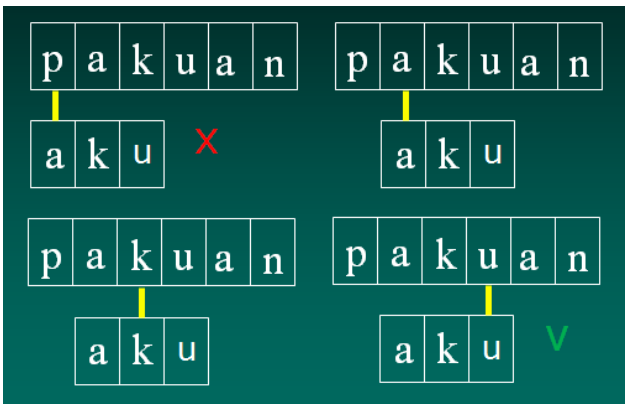
A. Algoritma *Pattern Matching*

Algoritma yang akan digunakan untuk menyelesaikan masalah pengecekan plagiarisme pada dua buah lagu adalah dengan algoritma *pattern matching*. Algoritma *pattern matching* adalah algoritma yang akan mencari suatu bagian kecil (*pattern*) di dalam suatu bagian yang besar. Algoritma string matching ini pun dapat menggunakan tiga algoritma yang berbeda, yaitu algoritma *brute force*, algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer Moore (BM).

B. Algoritma *Brute Force*

Brute Force adalah algoritma yang *straight forward* atau tidak dipikir panjang terlebih dahulu. Dan algoritma ini kebanyakan mengandalkan tenaga atau waktu daripada efisiensi. Akan tetapi algoritma *brute force* ini jugadapat memecahkan beberapa masalah, bahkan terdapat beberapa masalah yang hanya bisa diselesaikan hanya dengan algoritma *brute force*.

Brute force didalam algoritma *pattern matching* dilakukan dengan mengecek satu persatu pattern yang ada. Jika ditemukan tidak sama maka akan geser satu dan dilakukan terus sampai ujung yang bisa dicek. Berikut ilustrasi pengecekkan pattern dengan algoritma *brute force*.



Gambar 1. Ilustrasi penggunaan *brute force* dalam *string matching*

```
   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()
```

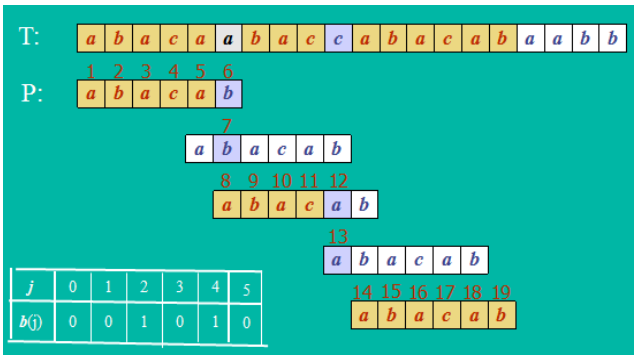
Gambar 2. Algoritma *brute force* dalam bahasa Java
(Sumber : Slide kuliah pencocokan string oleh Dr. Andrew Davison dan diperbaharui oleh Dr. Rinaldi Munir)

Tulisan di atas merupakan coding untuk algoritma *brute force*. Kompleksitas dari algoritma *brute force* dalam *pattern matching* adalah O(mn).

C. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt (KMP) adalah algoritma yang melihat pattern dari kiri ke kanan seperti algoritma *brute force*. Tetapi algoritma KMP menggeser patternnya jauh lebih cerdas daripada algoritma *brute force*. Algoritma KMP akan menggunakan fungsi pinggiran yang digunakan untuk menghitung berapa jauh pattern akan bergeser.

Nilai dari fungsi pinggiran ini akan dilihat dari banyaknya prefiks dari pattern ke 0 sampai n yang juga merupakan sufiks pattern ke 1 sampai n. Misal pattern "akua" maka akan memiliki nilai fungsi pinggiran bernilai satu, karena hanya memiliki satu prefiks dan juga prefiks yaitu "a", dan jika dilanjutkan akan menghasilkan "ak" dam "ua" maka tidak sama, sehingga nilainya adalah satu. Karena nilai fungsi pinggirannya bernilai satu, maka pattern akan digeser sejauh banyaknya pattern dikurangi nilai fungsi pinggiran yang ada. Sehingga dalam kasus ini akan digeser sejauh 4 -1 = 3 geser. Berikur ilustrasi *pattern matching* menggunakan algoritma KMP dengan memanfaatkan fungsi pinggiran.



Gambar 3. Ilustrasi penggunaan algoritma KMP dalam *string matching*
(Sumber : Slide kuliah pencocokan *string* oleh Dr. Andrew Davison dan diperbaharui oleh Dr. Rinaldi Munir)

Pada contoh diatas, diketahui *pattern*nya adalah "abacab" dengan fungsi pinggiran yang sudah dihitung. Kita coba hitung salah satu nilai fungsi pinggiran, yaitu pada j = 2, panjang *pattern* adalah tiga, dan *string*nya adalah "aba". Maka prefiks dari j=0 sampai j=2 dan juga sufiks dari j=1 sampai j=2 adalah a dan hanya memiliki panjang string satu. Maka dari itu nilai fungsi pinggiran pada j = 2 senilai satu.

Dalam ilustrasi tersebut dapat dilihat pada baris pertama *pattern* yang dipakai adalah j = 4, dengan fungsi pinggiran 1, sehingga akan digeser sebanyak 5 – 1 = 4 kali geser. Lalu di baris dua *pattern* yang dipakai adalah j = 0, dengan fungsi pinggiran 1, sehingga akan digeser sebanyak 1 – 0 = 1 kali. Pada baris ketiga digunakan *pattern* j = 3, dengan fungsi pinggiran 0 maka akan digeser sebanyak 4 – 0 = 4 kali. Lalu pada baris ke eempat, digunakan j = 0, yang akan menggeser sebanyak 1 kali. Di baris terakhir dicek, dan terbukti bahwa *pattern* ditemukan.

Dalam hal kompleksitas waktu, waktu yang dimiliki oleh KMP tentu sangat cepat jika dibandingkan dengan *brute force* yang mempunyai kompleksitas O(mn). Kompleksitas yang dari dimiliki oleh fungsi pinggiran dalam menghitung nilai pinggiran adalah O(m) sedangkan untuk pencarian string adalah O(n). Maka kompleksitas total waktu untuk algoritma KMP adalah O(m+n). Berikut adalah implementasi algoritma KMP dan juga fungsi pinggiran dalam bahasa 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
    } // end of kmpMatch()
```

Gambar 4. Algoritma KMP dalam bahasa Java
(Sumber : Slide kuliah pencocokan *string* oleh Dr. Andrew Davison dan diperbaharui oleh Dr. Rinaldi Munir)

```
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()
```

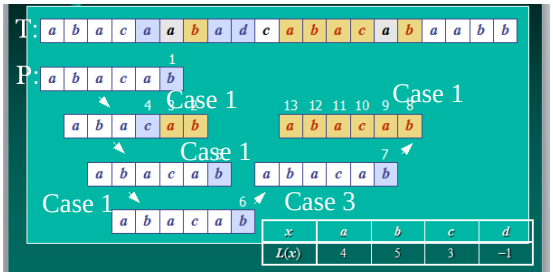
Gambar 5. Implementasi fungsi pinggiran dalam bahasa Java
(Sumber : Slide kuliah pencocokan *string* oleh Dr. Andrew Davison dan diperbaharui oleh Dr. Rinaldi Munir)

D. Algoritma Boyer – Moore (BM)

Algoritma Boyer – Moore adalah algoritma *pattern matching* yang didasari oleh dua teknik, pertama teknik *looking-glass* yaitu teknik yang pengecekannya dilakukan dari belakang pattern adn kedua teknik *character – jump* yaitu teknik melompat jika *character* tidak sama.

Dalam algoritma boyer – moore ini, terdapat tiga kasus untuk melompati karakternya. *Case* pertama adalah jika di dalam *pattern* mengandung x, maka geser *pattern* hingga x di dalam *pattern* dan di dalam T yang di cari sejajar. Case kedua adalah jika di dalam pattern mengandung x, tetapi tidak mungkin untuk menyejajarkan keduanya, maka pattern bergeser satu. Case ketiga adalah jika case dua dan case satu tidak berhasil, maka langsung lompati pattern hingga melewati x tadi.

Di dalam Boyer – Moore juga digunakan fungsi pendukung untuk memudahkan *character-jump*, fungsi tersebut dinamakan fungsi *last occurence*. Fungsi ini dilakukan ketika ditemukan case satu, sehingga digeser hingga sejajar dengan nilai dari fungsi *lasti occurence* ini. Fungsi *last occurence* dimiliki oleh tiap character yang berbeda dan jika ada dua karakter yang sama, maka nilai yang paling besarlah yang diambil. Berikut adalah ilustrasi penggunaan algoritma Boyer – Moore pada algoritma *pattern matching.*



Gambar 6. Ilustrasi penggunaan algoritma BM dan fungsi *last occurence*
(Sumber : Slide kuliah pencocokan *string* oleh Dr. Andrew Davison dan diperbaharui oleh Dr. Rinaldi Munir)

Dapat dilihat ilustrasi penggunaan algoritma BM dan fungsi *last occurence* yang dapat dilihat di bagian bawah gambar. Terdapat step step dan analisis case yang telah terlihat di gambar. Berikut penjelasan dan analisis pada salah satu step dan *case* yang diambil, contohnya adalah pada step satu. Dapat dilihat pada step satu merupakan case pertama, karena pada pengecekkan, ditemukan di teks berupa karakter a,

padahal pada pattern merupakan karakter b. Maka geser sampai karakter ke 4 (seperti yang dapat dilihat di dalam tabel *last occurence*, perlu diingat *pattern* dimulai dari karakter ke 0).

Algoritma Boyer – Moore memiliki kompleksitas yang hampir sama dengan algoritma KMP, akan tetapi jauh lebih cepat jika dibandingkan dengan algoritma *brute force*. Algoritma ini memiliki kompleksitas O(mn + A) dengan A merupakan banyaknya tipe karakter yang muncul. Akan tetapi dalam kondisi normal, kompleksitas algoritma ini adalah O(m + n) setara dengan KMP. Algoritma ini lemah jika digunakan pada string biner dan akan jauh lebih cepat jika patternnya memiliki tipe karakter yang variatif. Berikut adalah implementasi algoritma Boyer Moore dan fungsi *last occurence*.

```
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()
```

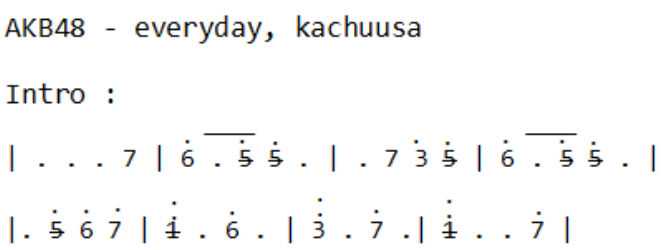
Gambar 7. Implemenatasi Algoritma Boyer – Moore dalam Bahasa JAVA

(Sumber : Slide kuliah pencocokan *string* oleh Dr. Andrew Davison dan diperbaharui oleh Dr. Rinaldi Munir)

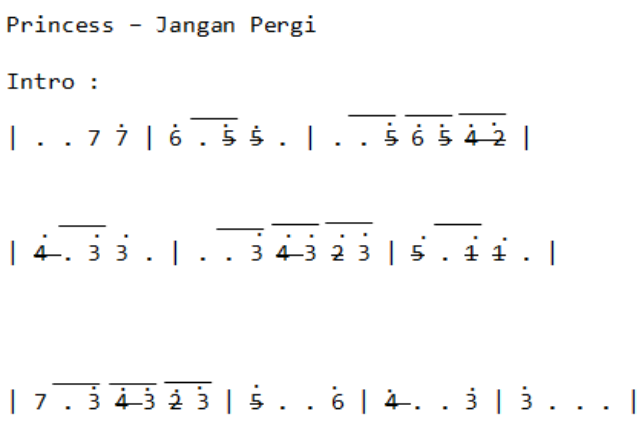
III. PENGECEKAN PLAGIARISME DENGAN PATTERN MATCHING

Telah disebutkan sebelumnya di Bab I tentang plagiarisme, bahwa suatu lagu dikatakan memplagiat lagu lain jika terdapat kesamaan sebanyak 8 bar. 8 bar ini dilihat dari not balok suatu lagu. Karena keterbatasan data yang bisa didapat, maka saat ini hanya digunakan not angka untuk referensi pengecekan plagiarisme. Sebenarnya not balok dan not angka tidak terlalu berbeda, hanya tampilannya saja yang berbeda, not balok pun bisa diterjemahkan menjadi not angka. Maka dari itu penggantian data dari not balok menjadi not angka bukan menjadi suatu masalah yang akan mengubah hasil analisis. Pada contoh kali ini, akan dicek 4 buah lagu yang bisa dijadikan data pengecekan plagiarisme. Lagu yang dicek adalah lagu AKB48 – *Everyday*, *Kachuusa* dengan Princess – Jangan Pergi dan Armada – Asal Kau Bahagia dengan F4 – Liu Xing Yu.

Pada perbandingan pertama terdapat dua lagu yaitu lagu AKB48 – Everyday, Kachuusa dengan Princess – Jangan Pergi. Berikut penggalan not balok dari kedua lagu tersebut.



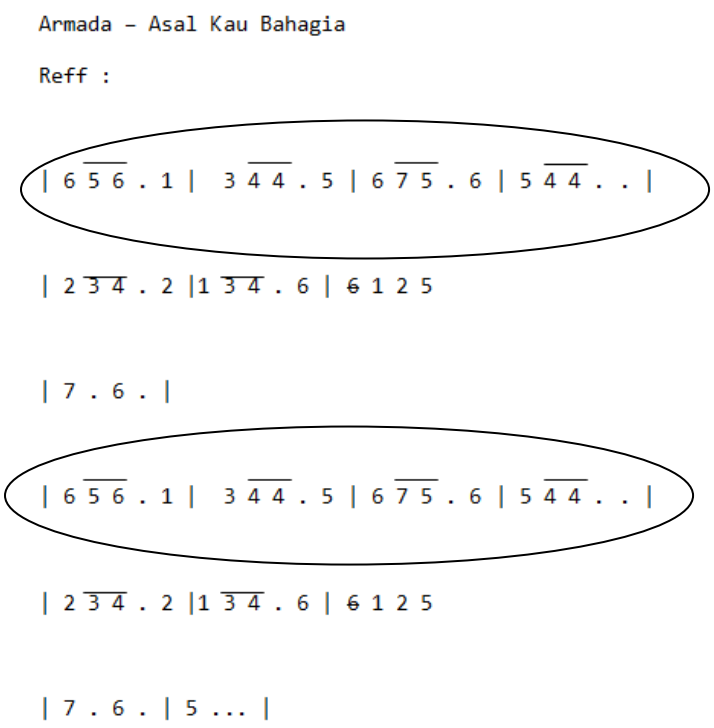
Gambar 8. Not Angka Lagu AKB48 – Everyday



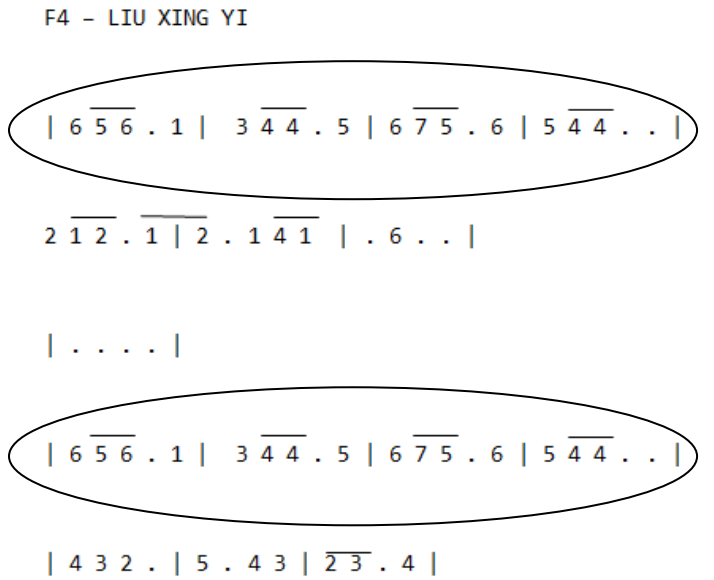
Gambar 9. Not Angka Lagu Princess – Jangan Pergi

Penggalan tersebut diambil karena bagian tersebut yang menurut beberapa orang terdengar sama. Namun jika dicocokkan kedua not balok tersebut terlihat berbeda. Walaupun ada satu bar yang sama persis dan beberapa bar yang memiliki pola yang sama. Kesimpulannya kedua lagu tersebut tidak bisa dikatakan plagiat. Pengambilan penggalan juga bisa dijadikan salah satu acuan heuristik.

Perbandingan kedua digunakan dua lagu yaitu Armada – Asal Kau Bahagia dan F4 – Liu Xing Yi. Berikut penggalan not balok dari kedua lagu tersebut.



Gambar 10. Not Angka Lagu Armada – Asal Kau Bahagia



Gambar 11. Not Angka F4 – Liu Xing Yi

Penggalan tersebut diambil pada bagian reff karena banyak orang yang mengatakan bahwa reff kedua lagi ini sama persis. Setelah diliat dari not balok nya, ternyata tidak sama persis, tetapi ada beberapa bar yang sama persis. Total 8 bar hanya pada satu reff saja. Dengan begitu dapat dikatakan bahwa kedua lagu ini sama dan melanggar Undang – Undan hak cipta, dan karena lagu dari F4 dikeluarkan terlebih dahulu, maka lagu Armada mungkin dikatakan plagiat.

Kedua contoh diatas masih menggunakan tangan dan mata manusia untuk mengenali nada dan mencocokkan nada. Jika diimplementasikan kedalam suatu aplikasi, banyak upgrade yang mungkin terjadi. Kepekaan nada akan lebih tajam, jika hanya di turunkan atau dinaikkan nada dasarnya saja bisa dideteksi dan dapat pula mengoreksi adanya perbedaan minor dan noise yang ada, sehingga hasil bisa jauh lebih teliti. Untuk pencocokkan patternnya, dilakukan per bar merupakan suatu karakter. Misal lagu pertama sebagai acuan, lagu kedua yang akan diiterasi. Ambil 8 bar pertama, lalu periksa dengan algoritma *pattern matching*. Jika terdapat perbedaan yang cukup besar dalam suatu bar, maka bar tersebut dinyatakan tidak sama. Begitu seterusnya hingga lagu habis. Jika belum ditemukan, iterasi lagi dengan 6 bar, 4 bar dan 2 bar. Lalu tunjukkan hasilnya terdapat pada menit keberapa, berapa banyak bar yang sama dan sebagainya. Jika total bar yang sama berjumlah 8, maka mungkin dikatakan kedua lagu yang dibandingkan sama dan salah satunya plagiat dari yang lainnya.

IV. KESIMPULAN

Plagiarisme sudah banyak dilakukan, bahkan beberapa orang tidak ragu untuk melakukan plagiarisme. Plagiarisme jawaban tugas misalnya. Mungkin jika plagiarisme jawaban tugas tidak terlalu terlihat merugikannya bagi orang lain, tetapi tidak dengan plagiarisme lagu. Jika lagu seseorang di plagiat, lalu yang lebih terkenal adalah lagu yang di plagiatkan, tentu akan menciptakan kerugian. Persoalan plagiarisme lagu juga sudah diatur dalam Undang Undang tentang hak cipta. Sering kali plagiarisme ini memunculkan ambigu, ada yang memang melakukan plagiarisme ada yang menepis dengan alasan terinspirasi. Dengan menggunakan algoritma pattern matching kita dapat membandingkan kedua lagu dan menentukan apakah salah satunya plagiat atau bukan. Tetapi seperti yang dituliksan di dalam Undang Undang, lagu dikatakan plagiat tidak hanya jika ada 8 bar yang sama, tetapi jika ada bagian yang familiar yang sama, maka dihitung plagiarisme. Untuk itu masih belum bisa dibuktikan dengan matematis, namun bisa menggunakan beberapa nilai heuristik

V. UCAPAN TERIMA KASIH

Puji syukur penulis panjatkan kepada Allah SWT, karena atas rahmat-Nya makalah ini dapat selesai. Penulis juga mengucapkan banyak terima kasih kepada Dr. Ir. Rinaldi Munir, selaku dosen pengajar mata kuliah Strategi Algoritma yang mengajar di kelas saya yaitu kelas K3, karena berkat ajaran beliau, penulis mendapatkan ilmu terkati mata kuliah Strategi Algoritma sehingga memungkinkan penulis untuk menulis makalah ini. Selanjutnya, penulis mengucapkan terima kasih kepada sumber referensi penulis dalam

menyelesaikan makalah ini dan semua pihak yang telah membantu penulis baik secara langsung maupun tidak langsung. Semoga dengan selesainya makalah ini, makalah ini dapat berguna bagi pembaca.

REFERENCES

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

[2] http://klikmusikid.tumblr.com/post/29325838203/klik-fakta-cara-tahu-musik-plagiat diakses pada Kamis, 18 Mei 2017 15:07

[3] https://syaifalmandiri.wordpress.com/2012/04/03/plagiat-dan-uu-tentang-plagiat/ diakses pada Kamis, 18 Mei 2017 15:00

[4] http://marciamusicportal.blogspot.co.id/2011/01/undang-undang-hak-cipta-tahun-2002.html diakses pada Kamis, 18 Mei 2017 15:09

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, 19 April 2017

Ega Rifqi Saputra (13515 015)