# Penerapan *Pattern Matching* untuk Deteksi Plagiarisme Tugas

Rifki Afina Putri (13511066)[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*[1]13511066@std.stei.itb.ac.id*

**Abstrak—Makalah ini berisi tentang *pattern matching* dan salah satu penerapannya dalam kehidupan sehari-hari. *Pattern matching* merupakan suatu pendekatan untuk mencari kecocokan pola pada suatu benda. Biasanya, pencocokan dilakukan untuk mencari suatu *pattern* dalam sebuah *string* atau teks. Banyak sekali penerapan *pattern matching* yang dapat kita aplikasikan dalam kehidupan sehari-hari. Salah satunya adalah penerapan *pattern matching* untuk mendeteksi plagiarisme dalam pengerjaan tugas. Plagiarisme dapat dideteksi dengan melakukan pencocokan *pattern* pada teks dokumen-dokumen tugas yang ada. Apabila tingkat kemiripan dokumen tersebut tinggi, terdapat kemungkinan adanya plagiarisme dalam dokumen tersebut.**

**Kata Kunci—Boyer-Moore, *Cosine Similarity*, KMP, *Pattern Matching*, Plagiarisme.**

## I. PENDAHULUAN

Perkembangan teknologi yang begitu pesat ternyata tidak selalu memberikan dampak yang positif bagi kita semua. Banyak juga dampak negatif yang muncul akibat adanya perkembangan teknologi ini. Misalnya saja, pesatnya perkembangan teknologi mengakibatkan maraknya plagiarisme. Menurut Kamus Besar Bahasa Indonesia, definisi plagiarisme adalah penjiplakan yang melanggar hak cipta. Penjiplakan ini sendiri dapat terjadi dimana saja, termasuk di kalangan mahasiswa, terutama dalam mengerjakan tugas.

Di era modern seperti ini, hanya dengan akses internet, kita bisa mendapatkan informasi yang kita inginkan dengan cepat dan mudah. Kemudahan dalam mencari informasi dan mudahnya akses internet secara tidak sadar mengubah pola pikir seseorang menjadi selalu berpikir instan dan praktis. Sayangnya, kemudahan dalam mencari informasi tersebut kadang disalahgunakan. Tidak jarang mahasiswa yang dengan mudahnya menjiplak hasil karya orang lain tanpa mencantumkan sumbernya. Mereka hanya berpikir praktis dan membuat tugas yang asal jadi dengan melakukan *copy paste* karya orang lain tanpa mencantumkan sumber kutipan tersebut pada tugas mereka.

Perilaku negatif seperti ini tentu berdampak buruk, bukan hanya merugikan pihak sumber yang membuat karya tersebut, perilaku ini juga merugikan mahasiswa yang melakukan plagiarisme itu sendiri. Oleh karena itu, maraknya bentuk plagiarisme ini perlu dicegah dan ditanggulangi, salah satunya dengan melakukan deteksi pada plagiarisme. Dengan adanya pendeteksi ini, diharapkan mahasiswa dapat berpikir dua kali untuk melakukan plagiarisme karena perilakunya dapat saja terdeteksi oleh program ini. Deteksi ini dilakukan dengan menggunakan konsep *pattern matching* yang diajarkan pada kuliah Strategi Algoritma. Dengan mencari kecocokan *pattern* pada teks dokumen tugas tersebut, diharapkan plagiarisme dapat terdeteksi.

## II. TEORI DASAR

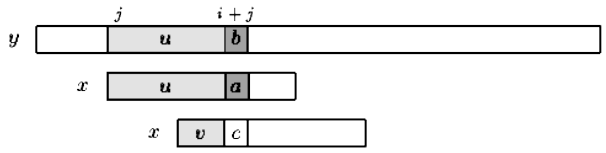
Algoritma *pattern matching* atau pencocokan *string* merupakan algoritma yang digunakan untuk mencari kecocokan suatu *pattern* di dalam sebuah *string* atau teks. Ada banyak metode pencocokan *string* yang sering digunakan. Pada makalah ini, digunakan dua metode algoritma pencocokan *string*, yaitu algoritma Knuth-Morris-Pratt (KMP) dan algoritma Boyer-Moore.

### *A. Algoritma Knuth-Morris-Pratt*

Algoritma Knuth-Morris-Pratt (KMP) adalah salah satu algoritma pencarian *string*, dikembangkan secara terpisah oleh Donald E. Knuth pada tahun 1967 dan James H. Morris bersamaVaughan R. Pratt pada tahun 1966, namun keduanya mempublikasikannya secara bersamaan pada tahun 1977.

Algoritma KMP merupakan algoritma yang mencari *pattern* dari kiri ke kanan, sama seperti pada algoritma *brute force*. Namun, pergeseran *pattern* ketika ditemukan ketidakcocokan pada algoritma ini lebih efisien apabila dibandingkan dengan algoritma *brute force*. Ide dasar dari algoritma ini adalah dengan menggeser sebanyak prefiks dari *pattern* P[1..j-1] yang merupakan sufiks dari *pattern* P[1..j-1]. Oleh karena itu, algoritma ini lebih mangkus apabila dibandingkan dengan algoritma *brute force* karena pada algoritma ini terdapat *array* atau tabel yang menentukan seberapa panjang kita harus menggeser *pattern* seandainya ketidakcocokan terjadi. Sedangkan

pada algoritma *brute force* pergeseran dilakukan satu per satu, yang tentu akan memakan waktu lebih lama.




**Gambar 1** Pergeseran *pattern* pada algoritma KMP (Sumber Gambar: http://www-igm.univ-mlv.fr/~lecroq/string/node8.html)

Dari gambar di atas, dapat terlihat bahwa pergeseran dilakukan sebanyak prefiks dari *pattern* x yang merupakan sufiks dari *pattern* x. Sehingga, tidak perlu lagi dilakukan perbandingan pada v. Perbandingan dilanjutkan dengan membandingkan karakter v + 1.

Pada KMP terdapat *border function* yang digunakan untuk menghitung banyaknya pergeseran. Misalnya saja, j adalah indeks posisi ketika terjadi ketidakcocokan pada *pattern* P, sedangkan k adalah indeks posisi sebelum ketidakcocokan (k = j-1). Maka, *border function* b(k) didefinisikan sebagai ukuran dari prefiks terbesar *pattern* P[1..k] yang merupakan sufiks dari *pattern* P[1..k]. Misalkan untuk b(5) dengan *pattern* P "abaaba", kita harus mencari prefiks terbesar dari "abaab" yang merupakan sufiks dari "abaab". String tersebut adalah "ab" dan ukurannya adalah 2. Oleh karena itu, besarnya b(5) adalah 2. Seperti telah disebutkan sebelumnya, pada kode, b(k) sendiri direpresentasikan sebagai *array*. (Dr. Andrew Davidson, 2006)

Berikut ini merupakan *pseudo code* untuk algoritma KMP dan *border function* KMP.

```
Input  : Pattern sebanyak m karakter
Output : Border function b untuk
P[i..j]
```

```
i ← 1
j ← 0
b(0) ← 0
while i < m do
    if P[j] = P[i] then
        b(i) ← j+1
        i ← i+1
        j ← j+1
    else if j > 0
        j ← b(j-1)
    else
        b(i) ← 0
        i ← i+1
return b
```

*Pseudo code* untuk *border function*

```
Input  : Teks dengan jumlah n karakter
(T[0..n])  dan  Pattern  jumlah  m
karakter (P[0..m])
Output : Indeks pertama dari substring
T yang cocok dengan P
```

```
f ← hitung border function pattern P
i ← 0
j ← 0
while i < length[T] do
    if j ← m-1 then
        return i-m+1  {match}
    i ← i+1
    j ← j+1
    else if j > 0
        j ← f(j-1)
    else
        i ← i+1
```

*Pseudo code* algoritma KMP

Algoritma KMP bagus digunakan untuk memproses file yang sangat besar, karena algoritma ini tidak pernah bergerak mundur ketika melakukan pencocokan pada teks. Selain itu, kompleksitas algoritma KMP sangat cepat apabila berjalan pada waktu yang optimal, yaitu sebesar O(m + n).

### *B. Algoritma Boyer-Moore*

Algoritma lain yang digunakan untuk mencari kecocokan *pattern* pada suatu *string* ialah algoritma Boyer-Moore. Algoritma ini dibuat oleh R.M Boyer dan J.S Moore. Ide utama dari algoritma Boyer-Moore adalah mencari *string* dengan melakukan pembandingan karakter mulai dari karakter paling kanan *pattern* dari *string* yang dicari. Dengan menggunakan algoritma ini, secara rata-rata proses pencarian akan menjadi lebih cepat jika dibandingkan dengan algoritma lainnya.

Pencocokan string dengan algoritma Boyer-Moore memiliki dua teknik:

1. Teknik *looking-glass*
   Merupakan teknik untuk mencari *pattern* P pada teks T dengan bergerak mundur, dimulai dari indeks terakhir pada P hingga indeks pertama pada P.
2. Teknik *character-jump*
   Merupakan teknik yang dilakukan ketika terjadi ketidakcocokan antara karakter *pattern* dan teks saat dilakukan pencocokan pada indeks tertentu.
   Terdapat tiga kasus yang mungkin terjadi pada teknik ini.
   Misalkan T[i] adalah array teks dengan indeks ke-i, dan P[j] adalah *array pattern* dengan indeks ke-j. Ketidakcocokan terjadi pada T[i] == x.
   - Jika pada *pattern* P terdapat karakter x pada posisi yang lain, maka geser P ke kanan sebanyak *last occurence* x.
   - Jika pada *pattern* P terdapat karakter x pada posisi yang lain, tetapi pergeseran menuju *last occurence* tidak memungkinkan, maka geser P ke kanan sebanyak satu karakter ke T[i+1].
   - Jika kasus pertama dan kedua tidak berlaku, maka geser P ke kanan hingga P[1] sejajar dengan T[i+1].

Pada algoritma Boyer-Moore, terdapat fungsi *last occurence* (dinotasikan sebagai L()). Fungsi ini digunakan untuk menyimpan posisi *last occurence* karakter A pada

pattern P. L(x) akan menghasilkan suatu integer, yaitu indeks terbesar i dimana P[i] == x atau -1 jika indeks tidak ada. (Dr. Andrew Davidson, 2006)

Berikut ini merupakan *pseudo code* untuk algoritma Boyer-Moore.

```
Input  : Teks dengan jumlah n karakter
(T[0..n]) dan Pattern jumlah m
karakter (P[0..m])
Output : Indeks pertama dari substring
T yang cocok dengan P
```

```
Hitung last occurence
i ← m-1
j ← m-1
repeat
    if P[j] = T[i] then
        if j=0 then
            return i   {match}
        else
            {teknik looking-glass}
            i ← i-1
            j ← j-1
    else
        {character jump}
        i ← i+m-Min(j,1+last[T[i]])
        j ← m-1
until i > n-1
return "no match"
```

*Pseudo code* algoritma Boyer-Moore

Waktu eksekusi *worst case* untuk algoritma Boyer-Moore adalah O(nm+A). Algoritma ini cepat ketika alfabet (A) bernilai banyak dan cenderung lambat apabila jumlah alfabetnya sedikit.

### *C. Cosine Similarity*

Selain menggunakan metode *pattern matching* untuk mencari *pattern* yang sesuai, diperlukan juga suatu metode untuk menghitung bobot kemiripan kata dalam suatu teks. Salah satu metode yang sering digunakan untuk menghitung kemiripan teks ialah metode *cosine similarity*. Pengukuran kemiripan pada metode ini dilakukan dengan menghitung nilai *cosinus* sudut antara dua vektor. Masing-masing teks direpresentasikan sebagai vektor.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

**Gambar 2** Rumus *cosine similarity*
(Sumber Gambar: https://www.bionicspirit.com/blog/2012/01/16/cosine-similarity-euclidean-distance.html)

A dan B merupakan vektor dari dua teks yang akan dibandingkan. Dari rumus di atas, jika perhitungan menghasilkan nilai -1, berarti teks sangat berlawanan. Apabila hasilnya 1, berarti teks sama persis. Apabila hasilnya 0, berarti teks merupakan teks yang independen.

## III. PEMBAHASAN

Secara umum, cara kerja yang dilakukan oleh program pendeteksi plagiarisme tugas ini ialah dengan melakukan perbandingan pada dua buah input teks yang dimasukkan oleh *user*. Untuk menyederhanakan persoalan, pada pengujian, format masukan teks berupa file berformat txt. Langkah-langkah yang dilakukan untuk menyelesaikan permasalahan deteksi plagiarisme ini dijelaskan secara lebih rinci pada poin-poin berikut.

- Dari dua file teks yang telah dimasukkan, dilakukan pengambilan informasi kata-kata apa saja yang terdapat pada teks tersebut. Masing-masing teks akan dipisahkan per kata dan kata-kata tersebut kemudian disimpan ke dalam sebuah tabel. Tabel ini berisi *pattern-pattern* yang akan dibandingkan nanti.
- Dilakukan perbandingan dengan menggunakan algoritma *string matching* yang telah dijelaskan sebelumnya (KMP dan Boyer-Moore). *String matching* dilakukan dengan cara melakukan perbandingan kata-kata pada tabel *pattern* dengan kedua teks. Perbandingan dilakukan satu per satu hingga seluruh isi tabel *pattern* telah dibandingkan dengan teks.
- Apabila seluruh *pattern* telah dibandingkan, selanjutnya dilakukan perhitungan masing-masing jumlah kemunculan *pattern* pada teks tersebut. Jumlah kemunculan *pattern* tersebut akan direpresentasikan menjadi suatu vektor. Sehingga, setiap file teks akhirnya akan memiliki representasi vektor sendiri.
- Dari dua vektor tersebut, kemudian dapat dilakukan perhitungan tingkat kemiripan dengan menggunakan rumus *cosine similarity*. Apabila hasil perhitungan telah didapat, selanjutnya dilakukan penarikan kesimpulan apakah teks tersebut mirip atau tidak. Jika tingkat kemiripan kedua teks tersebut tinggi, maka kemungkinan adanya plagiarisme pada teks tersebut juga tinggi.

### *A. Implementasi Menggunakan Algoritma KMP*

Pada bab sebelumnya, di bagian *pseudo code* telah dijelaskan bahwa algoritma KMP akan mengembalikan indeks pertama dari *substring* teks yang cocok dengan *pattern* P. Untuk memecahkan permasalahan deteksi plagiarisme ini, dilakukan sedikit modifikasi pada fungsi KMP sehingga keluaran fungsi bukan berupa indeks pertama dari *substring* teks yang cocok dengan *pattern* P, melainkan jumlah kemunculan *pattern* P di dalam teks. Modifikasi dilakukan dengan membuat fungsi menjadi rekursif. Perubahan fungsi menjadi rekursif dilakukan agar pencarian *pattern* tidak langsung berhenti ketika *pattern* pertama telah ditemukan pada teks. Dari *pseudo code* pada bab sebelumnya, dapat terlihat bahwa pencocokan akan langsung berhenti apabila *pattern* telah ditemukan pada suatu posisi teks. Padahal, kita ingin melakukan pencocokan pada keseluruhan *string*, bukan hanya pada

*substring* dari teks. Karena itulah, perubahan fungsi menjadi fungsi rekursif merupakan salah satu solusi yang bisa dilakukan.

Berikut ini merupakan *pseudo code* dari fungsi KMP yang telah dimodifikasi.

```
Input  : Teks dengan jumlah n karakter (T[0..n]) dan Pattern jumlah m karakter (P[0..m])
Output : Jumlah kemunculan pattern dalam suatu teks
```

```
f ← hitung border function pattern P
i ← 0
j ← 0
jumlah ← 0  {jumlah kemunculan}
index ← 0   {indeks ditemukannya pattern yang cocok}
while i < length[T] do
    if j ← m-1 then
        jumlah ← jumlah+1
        index ← i-m+1

        ubah masukan teks dengan
        menghapus substring yang
        ditemukan pada teks
        {rekursif}
        index ← KMP(masukan,pattern)
    i ← i+1
    j ← j+1
    else if j > 0
        j ← f(j-1)
    else
        i ← i+1

if index != -1 then
    return jumlah
else
    return 0
```

*Pseudo code* algoritma KMP yang telah dimodifikasi

Setelah dilakukan pencarian jumlah kemunculan, maka kata-kata yang mengandung *pattern* beserta jumlah kemunculannya akan dimasukan pada suatu tabel.

### *B. Implementasi Menggunakan Algoritma Boyer-Moore*

Sama halnya seperti pada algoritma KMP, pada algoritma Boyer-Moore juga perlu dilakukan modifikasi agar fungsi dapat mengembalikan jumlah kemunculan *pattern* pada teks. Seperti algoritma KMP, pada algoritma Boyer-Moore perubahan dilakukan dengan memodifikasi fungsi Boyer-Moore menjadi fungsi rekursif. Alasan-alasan mengapa perlu dilakukan modifikasi sama dengan yang telah dipaparkan sebelumnya pada bagian implementasi algoritma KMP. Letak perbedaan antara implementasi menggunakan algoritma KMP dan Boyer-Moore ini ialah perbedaan metode dalam melakukan pencarian dan pencocokan *pattern* pada teks. Umumnya, algoritma Boyer-Moore memiliki waktu eksekusi yang lebih cepat dibandingkan dengan waktu eksekusi pada algoritma KMP.

Berikut ini merupakan *pseudo code* dari fungsi Boyer-Moore yang telah dimodifikasi.

```
Input  : Teks dengan jumlah n karakter (T[0..n]) dan Pattern jumlah m karakter (P[0..m])
Output : Jumlah kemunculan pattern dalam suatu teks
```

```
Hitung last occurence
i ← m-1
j ← m-1
jumlah ← 0  {jumlah kemunculan}
index ← 0   {indeks ditemukannya pattern yang cocok}
repeat
    if P[j] = T[i] then
        if j=0 then
            jumlah ← jumlah+1
            index ← i

            ubah masukan teks dengan
            menghapus substring yang
            ditemukan pada teks
            {rekursif}
            index ← BM(masukan,pattern)
        else
            {teknik looking-glass}
            i ← i-1
            j ← j-1
    else
        {character jump}
        i ← i+m-Min(j,1+last[T[i]])
        j ← m-1
until i > n-1

if index != -1 then
    return jumlah
else
    return 0
```

*Pseudo code* algoritma Boyer-Moore yang telah dimodifikasi

Sama halnya dengan algoritma KMP, setelah dilakukan pencarian jumlah kemunculan, maka kata-kata yang mengandung *pattern* beserta jumlah kemunculannya akan dimasukan pada suatu tabel.

### *C. Implementasi Cosine Similarity*

Pada teknis program sendiri, setelah kemunculan *pattern* telah ditransformasikan menjadi vektor, selanjutnya dilakukan komputasi terhadap vektor tersebut menggunakan rumus *cosine similarity*. Seperti telah dijelaskan sebelumnya, rumus ini akan digunakan untuk menentukan bobot kemiripan diantara kedua teks (yang telah direpresentasikan sebagai vektor). Implementasi *cosine similarity* akan dijelaskan dalam bentuk *pseudo code*.

Terdapat tiga fungsi untuk mendapatkan *similarity*, yaitu fungsi untuk menghitung *dot product*, menghitung akar dari *dot product* vektor *(magnitude)*, dan fungsi untuk menghitung *similarity* itu sendiri. *Similarity*

merupakan fungsi *cosinus* yang didapat dengan membagi *dot product* dengan *magnitude* vektor.

```
Input  : vektor A (array of integer)
dan vektor B (array of integer)
Output  : Dot Product
dotProduct ← 0
for i=0 hingga i<panjang vektor A  do
    dotProduct ← dotProduct + (vekA[i]
    * vekB[i])
return dotProduct
```

*Pseudo code* untuk menghitung *dot product*

```
Input  : vektor (array of integer)
Output  : Magnitude
return sqrt(DotProduct(vektor,vektor))
```

*Pseudo code* untuk menghitung *magnitude*

```
Input  : vektor A (array of integer)
dan vektor B (array of integer)
Output  : similarity
dotProduct ← DotProduct(vekA, vekB)
magnitudeA ← Magnitude(vekA)
magnitudeB ← Magnitude(vekB)
similarity ← dotProduct/(magnitudeA *
magnitudeB)
return similarity
```

*Pseudo code* untuk menghitung *similarity*

### *D. Contoh Pengujian*

Data uji yang digunakan dalam pengujian deteksi plagiarisme ini adalah tiga buah file teks dengan isi sebagai berikut.

**Teks 1**: Kalimat ini merupakan kalimat untuk melakukan uji coba deteksi plagiarisme untuk tugas.
**Teks 2**: Kalimat yang saya buat ini, kalimat untuk melakukan uji deteksi plagiarisme tugas.
**Teks 3**: Kalimat yang ketiga sengaja dibuat sangat berbeda untuk memberikan contoh yang berbeda pula.

Seperti telah dijelaskan sebelumnya, terdapat empat langkah untuk menyelesaikan permasalahan ini.

Langkah pertama, dilakukan pengambilan informasi kata-kata apa saja yang akan dimasukkan ke dalam tabel *pattern*. Kata-kata tersebut bersumber dari isi ketiga file teks di atas. Setelah dilakukan penelusuran terhadap tiga teks, maka list kata-kata yang termasuk ke dalam tabel *pattern* adalah:
Kalimat, ini, merupakan, untuk, melakukan, uji, coba, deteksi, plagiarisme, tugas, yang, saya, buat, ketiga, sengaja, sangat, berbeda, memberikan, contoh, pula.

Pengambilan kata-kata di atas dilakukan dengan tidak mempertimbangkan kasus-kasus khusus dan unik yang mungkin terjadi.

Langkah kedua, pencocokan *pattern* terhadap masing-masing teks tersebut. Pencarian dapat dilakukan dengan menggunakan algoritma KMP dan Boyer-Moore. Baik menggunakan KMP ataupun Boyer-Moore, hasil pencariannya akan menghasilkan jumlah yang sama, hanya waktu eksekusi pecocokannya saja yang berbeda. Hasil pencocokan *pattern* akan mengembalikan jumlah kemunculan *pattern* pada teks yang hasilnya dirangkum dalam tabel berikut.

**Tabel 1** Hasil pencocokan *pattern*

| Kata | Jumlah Kemunculan | | |
|---|---|---|---|
| | Teks 1 | Teks 2 | Teks 3 |
| Kalimat | 2 | 2 | 1 |
| Ini | 1 | 1 | 0 |
| Merupakan | 1 | 0 | 0 |
| Untuk | 2 | 1 | 1 |
| Melakukan | 1 | 1 | 0 |
| Uji | 1 | 1 | 0 |
| Coba | 1 | 0 | 0 |
| Deteksi | 1 | 1 | 0 |
| Plagiarisme | 1 | 1 | 0 |
| Tugas | 1 | 1 | 0 |
| Yang | 0 | 1 | 2 |
| Saya | 0 | 1 | 0 |
| Buat | 0 | 1 | 1 |
| Ketiga | 0 | 0 | 1 |
| Sengaja | 0 | 0 | 1 |
| Sangat | 0 | 0 | 1 |
| Berbeda | 0 | 0 | 2 |
| Memberikan | 0 | 0 | 1 |
| Contoh | 0 | 0 | 1 |
| Pula | 0 | 0 | 1 |

Langkah ketiga, dari hasil di atas, dilakukan transformasi jumlah kemunculan *pattern* pada setiap teks ke dalam vektor. Setiap angka dari vektor merepresentasikan jumlah kemunculan dari setiap kata pada teks tersebut. Berikut ini hasil transformasi jumlah kemunculan setiap *pattern* pada teks.

Vektor teks 1: [2 1 1 2 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0]
Vektor teks 2: [2 1 0 1 1 1 0 1 1 1 1 1 1 0 0 0 0 0 0 0]
Vektor teks 3: [1 0 0 1 0 0 0 0 0 0 2 0 1 1 1 1 2 1 1 1]

Langkah keempat, dilakukan perhitungan bobot kemiripan teks antara dua teks (dua vektor) dengan menggunakan rumus *cosine similarity*. Berikut ini merupakan hasil perhitungan untuk setiap perbandingan teks.

Similarity teks 1 & 2 = 0.801783725737
Similarity teks 1 & 3 = 0.242535625036
Similarity teks 2 & 3 = 0.388922234131

Dari hasil pengujian, dapat terlihat bahwa teks 1 dan teks 2 memiliki bobot kemiripan yang tinggi (mendekati nilai 1) sehingga terdapat indikasi bahwa teks 1 merupakan plagiasi dari teks 2 ataupun sebaliknya. Kemudian, bobot kemiripan teks 3 dengan teks 1 maupun teks 2 cukup rendah, sehingga dapat disimpulkan bahwa tidak terdapat unsur plagiasi pada teks 3, karena teks 3 berbeda diantara teks yang lainnya.

## IV. KESIMPULAN DAN SARAN

Setelah dilakukan implementasi dan pengujian, dapat disimpulkan bahwa konsep *pattern matching* dapat diterapkan untuk mendeteksi plagiarisme pada teks, khususnya untuk deteksi plagiarisme tugas. Dalam tahapan deteksi plagiarisme itu sendiri, algoritma *pattern matching* berperan penting untuk mencari kecocokan teks dengan *pattern* yang telah didefinisikan dan menghitung jumlah kemunculan *pattern* tersebut. Hasil dari pencocokan *string* tersebut kemudian diolah untuk kemudian dihitung bobot kemiripan. Metode yang digunakan untuk menghitung bobot kemiripan pada makalah ini ialah metode *cosine similarity*. Pendekatan yang dilakukan pada makalah ini merupakan pendekatan yang cukup praktis dan mudah diimplementasikan. Namun, masih terdapat beberapa kekurangan pada metode pendekatan yang dilakukan untuk memecahkan persoalan deteksi plagiarisme ini.

Untuk saran pengembangan selanjutnya, perlu dilakukan analisis lebih lanjut algoritma *pattern matching* yang seperti apakah yang efektif digunakan dalam deteksi plagiarisme ini. Algoritma KMP dan Boyer-Moore yang digunakan pada makalah ini sendiri dirasa kurang efektif. Hal ini disebabkan oleh pencocokan *string* yang diimplementasikan pada kedua algoritma ini merupakan pencocokan *exact matching*. Artinya, *pattern* yang ada pada teks harus sama persis dengan *pattern* yang sedang dibandingkan. Sehingga bisa saja terjadi kesalahan perhitungan jumlah kemunculan untuk contoh uji kasus yang lebih rumit atau untuk kasus-kasus unik.

Solusi lain untuk mengatasi permasalahan ini ialah dengan melakukan ekstraksi terlebih dahulu ketika tahap pengambilan informasi untuk memasukkan kata pada tabel *pattern*. Maksud dari ekstraksi ini ialah setiap kata yang ada pada tabel *pattern* harus merupakan kata dasar. Pada uji coba kali ini, masih belum ada pembeda antara kata dasar dan kata yang berimbuhan. Selain pada tabel *pattern*, ekstraksi juga dapat dilakukan pada teks itu sendiri. Sebelum dilakukan pencocokan *string*, terlebih dahulu dilakukan perubahan seluruh kata pada teks menjadi kata dasar. Dengan melakukan ekstraksi tersebut, diharapkan pencocokan *string* dan perhitungan dapat lebih akurat. Selain itu, lebih baik lagi jika dilakukan penanganan untuk kasus-kasus unik, seperti penggunaan sinonim, penanganan jika terdapat kata ‘tidak’ dan masih banyak kasus unik lainnya.

Kemudian, pada uji coba kali ini, input yang dapat dimasukkan oleh *user* masih terbatas untuk file dengan format txt. Untuk pengembangan selanjutnya, diharapkan program dapat menerima input file dengan format lainnya seperti doc, pdf, dan lain-lain.

## REFERENSI


[1] Singhal, Amit (2001). "Modern Information Retrieval: A Brief Overview". Bulletin of the IEEE Computer Society Technical Committee on Data Engineering 24 (4): 35–43.

[2] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/kuthMP.htm (Tanggal akses: 19 Desember 2013 pukul 7.47)

[3] http://komputasi.files.wordpress.com/2011/11/ilustrasi-preprocessing-searching-dalam-text-mining.pdf (Tanggal akses: 19 Desember 2013 pukul 15.05)

[4] https://www.bionicspirit.com/blog/2012/01/16/cosine-similarity-euclidean-distance.html (Tanggal akses: 20 Desember 2013 pukul 3.52)

[5] Slide kuliah Strategi Algoritma dengan Topik *Pattern Matching.*


## 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, 20 Desember 2013

Rifki Afina Putri (13511066)