# Mencari Pola dalam Gambar dengan Algoritma *Pattern Matching*

Muhammad Farhan Majid (13514029)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132 Indonesia
13514029@std.stei.itb.ac.id

***Abstraksi*—Algoritma *pattern matching* (pencocokan pola) telah banyak digunakan dalam kehidupan manusia di era digital. Meskipun biasanya *pattern matching* digunakan untuk mencari potongan kalimat di dalam *string*, prinsip *pattern matching* itu sendiri dapat diaplikasikan ke dalam banyak persoalan. Salah satunya adalah pencarian pola pada gambar. Dengan menggunakan prinsip yang sama, kita dapat menemukan sebuah potongan gambar pada gambar aslinya.**

***Kata Kunci—pattern matching; gambar digital; brute force; Knuth-Morris-Pratt algorithm; Boyer-Moore algorithm***

## I. PENDAHULUAN

Gambar adalah media visual yang sudah tidak asing digunakan oleh manusia. Kehidupan manusia sejak zaman prasejarah tidak dapat dilepaskan dari gambar. Mulai dari gambar simbol-simbol pada dinding gua prasejarah, lukisan para pelukis *Renaissance* Eropa, hingga cetak biru arsitektur gedung pencakar langit saat ini. Keberadaan gambar tidak dapat dilepaskan dari sejarah peradaban manusia. Dengan berkembangnya teknologi dalam bidang komputer, gambar tidak lagi hanya dituangkan menggunakan kertas dan pensil, melainkan juga dalam bentuk digital.

Gambar digital direpresentasikan sebagai kumpulan titik dengan warna tertentu yang disusun sedemikian rupa sehingga membentuk sebuah representasi visual yang dimengerti manusia. Misalkan kumpulan titik berwarna hitam yang berjajar lurus dapat manusia artikan sebagai sebuah garis. Di sisi lain, komputer tidak memiliki kemampuan untuk mengidentifikasi 'keberadaan' garis tersebut, melainkan hanya sebatas susunan warna pada gambar digital tersebut.

Representasi warna yang dimiliki oleh gambar digital inilah yang menjadi menjadi bahasan utama tulisan ini. Dalam keseharian, sering kali kita ingin menemukan sebuah benda di dalam sebuah gambar digital. Misalkan kita ingin mencari seekor harimau di dalam gambar hutan. Dengan algoritma pencocokan pola (*pattern matching*), kita dapat mencari pola warna yang terdapat di dalam gambar untuk menemukan benda yang kita inginkan.

## II. DASAR TEORI

### A. Algoritma *Pattern Matching*

*Pattern matching* adalah sebuah proses pencarian pola dalam sekumpulan *token* yang berurutan. Pada umumnya, *pattern matching* digunakan untuk mencari pola karakter alfabet dalam sebuah kalimat (*string*).

Persoalan *pattern matching* dirumuskan sebagai berikut:

Diberikan sebuah teks (*text*), yakni *string* yang memiliki panjang *n* karakter; dan sebuah pola (*pattern*), yakni string dengan panjang *m* karakter ($m < n$) yang akan dicari di dalam *text*. Carilah kemunculan pertama *pattern* di dalam *text*.

Perhatikan ilustrasi berikut:

*Text*: `With gr`**`eat`**` power, comes great bills.`

*Pattern*: `eat`

*Pattern matching* akan mencari kemunculan pertama (dari kiri) *pattern* pada *text*. Dalam kasus ini, ditemukan *pattern* 'eat' pada karakter ke-8 *text* (spasi termasuk sebuah karakter/*token*).

Terdapat beberapa pendekatan yang digunakan dalam implementasi *pattern matching* pada program komputer, di antaranya dengan menggunakan *brute force*, *Knuth-Morris-Pratt* (*KMP*) *Algorithm*, dan *Boyer-Moore Algorithm*. Pendekatan-pendekatan ini dibedakan berdasarkan kompleksitas algoritma yang dihasilkan oleh masing-masing pendekatan.

#### Algoritma *Brute Force*

Pada pendekatan ini, setiap karakter pada *pattern* dicocokkan dengan karakter pada *text* secara *straightforward* (lempang). Jika diketahui text $T[1..n]$ dan pattern $P[1..m]$ adalah *array of character* dengan panjang masing-masing *n* dan *m* karakter, maka langkah penyelesaian dengan algoritma *brute force* adalah:

1. Sejajarkan *pattern* dan *text* pada awal karakter ($P[i]$ dan $T[j]$) untuk dibandingkan, dimana $i = j = 1$.
2. Bergerak dari kiri ke kanan, bandingkan setiap karakter pada *pattern* dengan karakter pada *text*. Perbandingan dilakukan dengan membandingkan $P[i]$ dan $T[j]$, kemudian $P[i+1]$ dan $T[j+1]$, dst. sampai ditemukan:
    a. semua karakter pada *pattern* ditemukan (pencarian selesai), atau

b. terdapat *mismatch* (ketidakcocokan) karakter.

3. Jika terdapat *mismatch* dan *text* belum habis, ulangi langkah 2 dengan $j = j+1$ (menggeser *pattern* sebanyak 1 karakter)

Berikut ilustrasi algoritma *brute force*:

*Text*: `Bear eats.`
*Pattern*: `eat`

```
  Bear eats.
1 eat
2  eat
3   eat
4    eat
5     eat
6      eat
```

*Pattern* 'eat' ditemukan pada karakter ke-6 *text* dengan 10 perbandingan karakter.

Jika menghitung jumlah operasi perbandingan yang dilakukan, kompleksitas waktu terbaik algoritma *brute force* adalah $O(n)$. Pada kasus terburuk, dilakukan sebanyak $m(n-m+1)$ perbandingan karakter.

### *Knuth-Morris-Pratt (KMP) Algorithm*

Pada algoritma *brute force*, jika terjadi *mismatch*, maka dilakukan pergeseran *pattern* sebanyak 1 karakter. Sedangkan pada algoritma *Knuth-Morris-Pratt* (*KMP*), setiap terjadi *mismatch*, pergeseran dilakukan berdasarkan informasi *pattern* (*longest-prefix-suffix* atau *border function*) yang telah diproses sebelum pencarian dimulai. Dengan informasi ini, jumlah pergeseran karakter dilakukan sesuai dengan informasi *pattern* yang dimiliki.

Berikut ilustrasi algoritma *KMP*:

*Text*: `dead deadpool.`
*Pattern*: `deadpool`

```
  dead deadpool.
1 deadpool
2    deadpool
3     deadpool
4      deadpool
```

*Pattern* 'deadpool' ditemukan pada karakter ke-6 *text* dengan 15 perbandingan karakter. Jika menggunakan *brute force*, diperlukan sebanyak 17 perbandingan karakter.

Kompleksitas waktu untuk memproses *border function* adalah $O(m)$, sedangkan pencarian pada *text* membutuhkan waktu $O(n)$. Maka, kompleksitas waktu algoritma *KMP* adalah $O(m+n)$.

### *Boyer-Moore Algorithm*

Algoritma *Boyer-Moore* didasarkan pada dua teknik, yakni:

1. *The looking glass technique*, yakni membandingkan karakter pada *pattern* dan *text* dimulai dari belakang-ke-depan (dimulai dari karakter terakhir *pattern*)
2. *The character-jump technique*, yakni jika terjadi *mismatch* pada karakter $x$ di *text* $T[i]$ dengan karakter $y$ di *pattern* $P[j]$, maka ada 3 kasus yang dicoba secara berurutan, yakni:
   a. Jika $P$ berisi karakter $x$ di kiri dari lokasi *mismatch*, maka sejajarkan $x$ dengan kemunculan terakhir $x$ pada *pattern*.
   b. Jika $P$ berisi karakter $x$ hanya di kanan dari lokasi *mismatch*, maka geser *pattern* sebanyak 1 karakter ke kanan.
   c. Jika $P$ tidak berisi karakter $x$, maka sejajarkan $P[1]$ dengan $T[i+1]$.

Berikut ilustrasi algoritma *Boyer-Moore*:

*Text*: `dead deadly deadpool.`
*Pattern*: `deadpool`

```
  dead deadly deadpool.
1 deadpool
2      deadpool
3          deadpool
4             deadpool
```

*Pattern* 'deadpool' ditemukan pada karakter ke-13 *text* dengan 11 perbandingan karakter.

Kompleksitas waktu terburuk algoritma *Boyer-Moore* adalah $O(mn+A)$, dimana $A$ adalah waktu yang dibutuhkan untuk melakukan *pre-processing* alfabet pada *pattern*.[2]

## *B.* Gambar *Digital*

Gambar *digital* (*digital image*) adalah kumpulan data komputer yang merepresentasikan sebuah gambar 2 dimensi. Gambar digital terbagi menjadi 2 kelompok, yakni *raster image* dan *vector image*. Pada tulisan ini, hanya *raster image* yang akan dibahas.

*Raster image* direpresentasikan sebagai *file* yang tersusun dari kumpulan *pixel* atau *picture element*. *Pixel*, sebagai satuan terkecil dari *raster image*, berisikan sebuah nilai yang merepresentasikan sebuah warna tertentu.[3]

# III. DESKRIPSI MASALAH

*Pattern matching* pada umumnya digunakan untuk mencari pola karakter pada *string*. Akan tetapi, prinsip algoritma ini dapat juga digunakan untuk mencari pola pada gambar *digital*.

Pada tulisan ini, penulis melakukan implementasi algoritma *pattern matching* untuk mencari pola (potongan gambar) pada gambar aslinya. Dalam persoalan ini, potongan gambar berperan sebagai *pattern* dan gambar digital berperan sebagai *text* (**Gambar 1**). Sedangkan *token* yang digunakan adalah *pixel*.

*Gambar 1 (kiri) Potongan gambar yang berperan sebagai pattern; dan (kanan) gambar aslinya yang berperan sebagai text [3]*

Proses pencarian dilakukan dengan membandingkan *pixel* (*token*) pada potongan gambar (*pattern*) dan gambar digital aslinya (*text*). Yang akan dicari pada persoalan ini adalah perbandingan dari 3 pendekatan *pattern matching*, yakni *brute force*, *KMP*, dan *Boyer-Moore*, dalam menemukan solusi. Yang dibandingkan adalah waktu pemrosesan, jumlah perbandingan karakter yang dilakukan, dan apakah hasil yang diperoleh sama atau tidak.

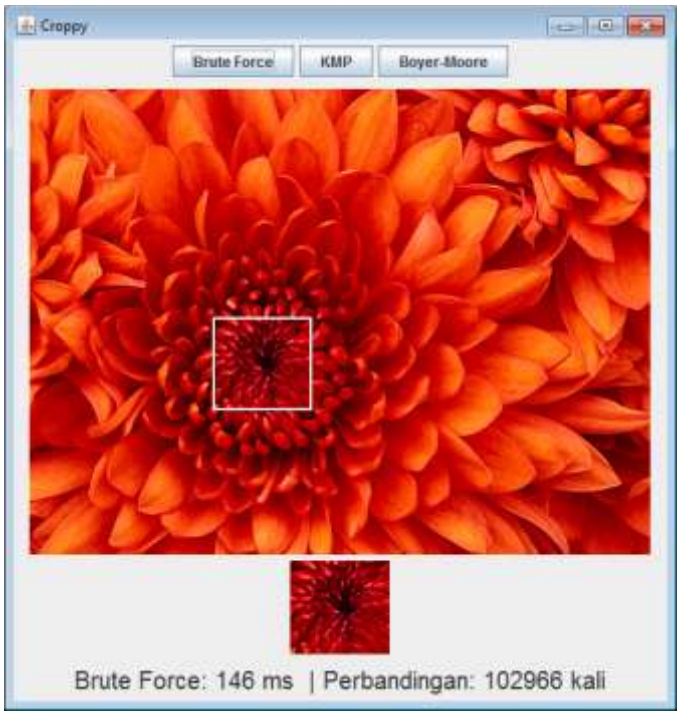
## IV. Implementasi dan Analisis Pengujian

### Implementasi

Penulis mengimplementasikan pemecahan masalah ke dalam sebuah program *Java* sederhana yang menerima masukan 2 *file* gambar, yakni potongan gambar dan gambar aslinya. Kemudian, program akan memproses kedua gambar untuk menemukan lokasi potongan gambar pada gambar aslinya.

Jika potongan gambar ditemukan, maka program akan menandai lokasi kemunculan potongan gambar pada gambar asli. Jika potongan gambar tidak ditemukan, maka program akan menampilkan pesan ke layar. Pencarian potongan gambar dilakukan 3 kali dengan 3 pendekatan yang hendak diuji, yakni *brute force*, *KMP*, dan *Boyer-Moore*. Untuk setiap pendekatan, dihitung waktu pemrosesannya dan jumlah perbandingan yang dilakukan sampai menemukan solusi.

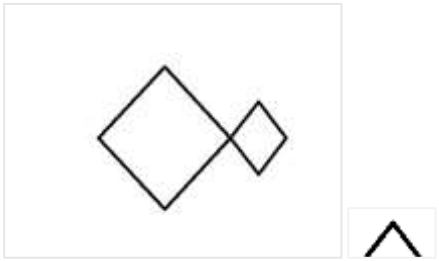
**Gambar 2** mengilustrasikan antarmuka program setelah pencarian. Setelah pengguna memberikan masukan berupa *text* dan *pattern*, pengguna dapat memilih algoritma apa yang ingin digunakan. Kemudian, program akan mencari *pattern* di dalam *text* dan menampilkan waktu eksekusi dan jumlah perbandingan karakter yang dilakukan. Jika *pattern* terdapat di dalam *text*, antamuka program akan menandai lokasi kemunculan *pattern* pada *text* (ditandai dengan kotak putih pada **Gambar 2**).



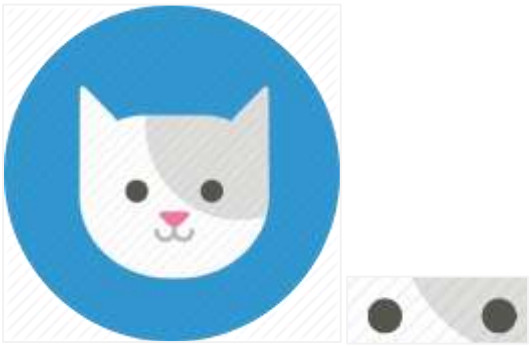

*Gambar 2 Antarmuka program pencarian pola dalam gambar menggunakan pattern matching*

Pengujian dilakukan terhadap beberapa masukan yang bervariasi dan diharapkan dapat menghasilkan informasi yang lengkap untuk dianalisis. Berikut adalah data uji (masukan) yang akan diberikan:

*Gambar 3 Data uji 1 kiri: text kanan: pattern[4]*



*Gambar 4 Data uji 2 kiri: text kanan: pattern*



*Gambar 5 Data uji 3 kiri: text kanan: pattern[5]*

*Gambar 6 Data uji 4 kiri: text kanan: pattern[3]*

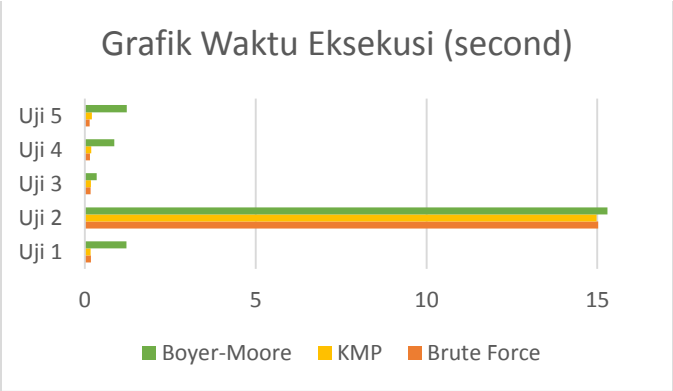
*Gambar 7 Data uji 5 kiri: text kanan: pattern[3]*

Berikut hasil pengujian program yang dilakukan:

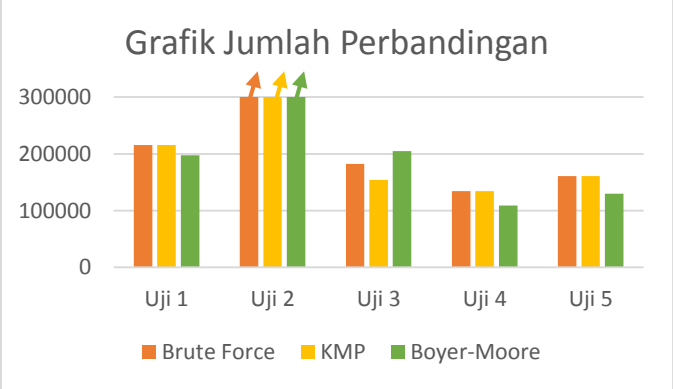
| Data ke- | Waktu eksekusi (*ms*) | | | Jumlah perbandingan | | |
|---|---|---|---|---|---|---|
| | BF | KMP | BM | BF | KMP | BM |
| 1 | 179 | 174 | 1219 | 215790 | 215790 | 197742 |
| 2 | 15025 | 14978 | 15298 | 51381509 | 51381509 | 51318050 |
| 3 | 170 | 179 | 346 | 182280 | 154110 | 205012 |
| 4 | 152 | 188 | 867 | 134347 | 134347 | 109271 |
| 5 | 141 | 204 | 1232 | 160995 | 160995 | 130189 |

*Tabel 1 Hasil pengujian program dengan brute force (BF), KMP, dan Boyer-Moore (BM)*

Pada saat pengujian, semua algoritma menghasilkan hasil yang sama (menemukan *pattern* pada lokasi yang sama pada *text*) dengan waktu eksekusi dan jumlah perbandingan karakter yang bervariasi. **Tabel 1** memperlihatkan hasil pengujian secara rinci. Data pada **Tabel 1** divisualisasikan pada **Gambar 8** dan **Gambar 9**.




*Gambar 8 Grafik perbandingan waktu eksekusi algoritma hasil pengujian*




*Gambar 9 Grafik perbandingan jumlah perbandingan karakter algoritma hasil pengujian*

## Analisis Pengujian

Berdasarkan hasil pengujian yang telah dilakukan, penulis berusaha membuat sebuah analisis yang juga didasarkan dari implementasi algoritma pada program *Java* yang dibuat. Ada beberapa poin yang penulis dapatkan dari hasil pengujian, di antaranya:

1. Waktu eksekusi

Berdasarkan **Gambar 8**, dapat diperhatikan bahwa algoritma *Boyer-Moore* selalu memakan waktu lebih lama dibandingkan dua algoritma sisanya. Ada faktor utama yang menjadi penyebabnya, yakni karena algoritma *Boyer-Moore* melakukan *pre-processing* alfabet pada *pattern* yang membutuhkan waktu $O(n)$, dimana $n$ adalah jumlah karakter pada *text*. Akan tetapi, dapat diperhatikan bahwa pada pengujian ke-2 algoritma *Boyer-Moore* memiliki waktu yang relatif sama dengan algoritma lainnya. Maka, wajar jika kita dapat menyimpulkan bahwa algoritma *Boyer-Moore* bekerja lebih optimal jika melakukan pencarian dengan iterasi yang banyak (bahkan pada pengujian ke-2, jumlah perbandingan algoritma *Boyer-Moore* jauh lebih baik dibandingkan algoritma lainnya). Pada pengujian lainnya yang iterasinya relatif sedikit, kinerja algoritma *Boyer-Moore* akan terbebani oleh *pre-processing* alfabet.

Algoritma *brute force* dan *KMP* memiliki waktu eksekusi yang relatif sama. Pada mayoritas kasus, algoritma *brute force* relatif lebih cepat dibandingkan *KMP*. Faktor utama yang menyebabkan hal ini adalah pemrosesan *border function*/*longest-prefix-suffix* yang dilakukan *KMP*. Meskipun tidak memakan waktu seperti *pre-processing* alfabet algoritma *Boyer-Moore*, pemrosesan *border function* cukup memperlambat waktu eksekusi *KMP*, terutama untuk pengujian dengan iterasi yang sedikit.

Pada pengujian ini, masukan yang digunakan (*text* dan *pattern*) memiliki ukuran yang relatif sama. Akan tetapi, seperti dapat diperhatikan pada **Gambar 8**, pengujian ke-2 memerlukan waktu pemrosesan yang jauh melebihi pengujian lainnya. Hal ini disebabkan oleh variasi *pixel* yang dimiliki oleh setiap gambar uji. Perhatikan **Gambar 4**. **Gambar 4** hanya terdiri dari 2 nilai *pixel*, yakni hitam(*RGB*:0,0,0) dan putih(*RGB*:255,255,255). Gambar seperti inilah yang bisa kita anggap sebagai kasus terburuk pencarian (*worst case*). Pada setiap algoritma, *mismatch* terjadi pada karakter-karakter yang posisinya 'jauh' dari awal *pattern*, sehingga algoritma membutuhkan waktu yang lama untuk bisa menemukan solusi. Hal yang sama tidak terjadi pada gambar uji sisanya, karena mereka memiliki nilai *pixel* yang bervariasi. Dengan begitu, algoritma akan lebih cepat menemukan *mismatch* pada *pattern* dan lebih cepat bergeser menemukan solusi.

2. Jumlah perbandingan

Berdasarkan **Gambar 9**, ada beberapa hal menarik yang dapat diperhatikan. Yang pertama adalah bahwa jumlah perbandingan algoritma *KMP* selalu sama dengan atau lebih kecil daripada algoritma *brute force*. Hal ini dikarenakan prinsip pencariannya yang hampir sama persis, yakni mencocokkan karakter dari kiri ke kanan. Yang membedakannya adalah jumlah pergeseran yang dilakukan setiap terjadi *mismatch*. Dari 5 kasus uji, 4 diantaranya menghasilkan jumlah perbandingan yang sama antara algoritma *brute force* dan *KMP*. Hal ini dikarenakan selalu terjadinya *mismatch* pada awal *pattern* yang menyebabkan tidak efektifnya *border function* yang dimiliki *KMP*.

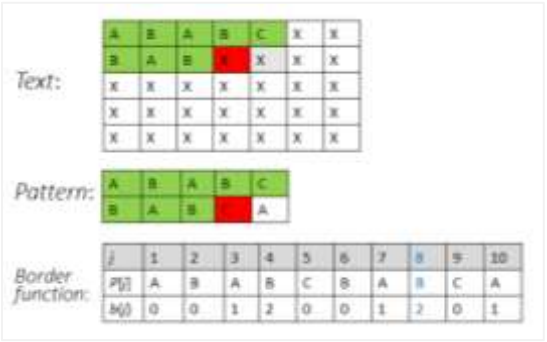
Kebalikan dengan *KMP*, algoritma *Boyer-Moore* unggul dari sisi jumlah perbandingan pada 4 kasus uji. Hal ini disebabkan oleh *character-jump technique* yang dimiliki algoritma *Boyer-Moore*. Dengan informasi yang dimiliki melalui *pre-processing* alfabet, algoritma *Boyer-Moore* mampu 'melompati' karakter lebih banyak dibandingkan algoritma lainnya.

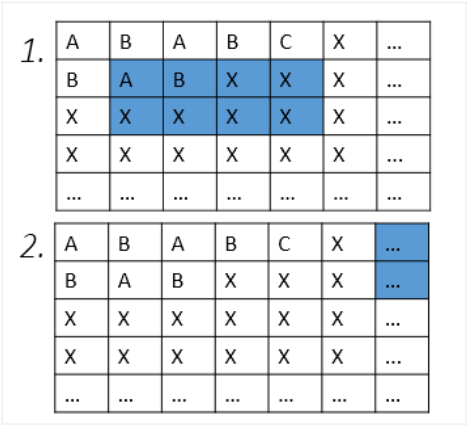
Seperti yang telah dijelaskan sebelumnya, kasus uji 2 memiliki jumlah perbandingan yang jauh lebih banyak dibandingkan kasus uji lain dikarenakan lokasi *mismatch* yang berada 'jauh' dari awal *pattern*.

3. Masalah implementasi algoritma *KMP* dan *Boyer-Moore* pada pencarian pola dalam gambar

Secara logika, *pattern matching* dapat diimplementasikan dengan mudah dalam pencarian *exact match* (sama persis) sebuah pola pada gambar aslinya. Hanya perlu mentranslasikan algoritma yang telah ada dan menggantikan 3 elemen utama dari algoritma ini, yakni mengganti *pattern string* dengan potongan gambar, mengganti *text string* dengan gambar aslinya, dan mengganti karakter (*token*) dengan *pixel* gambar.

Hal ini dapat diimplementasikan dengan mudah menggunakan pendekatan *brute force*. Akan tetapi, tidak jika menggunakan pendekatan lainnya, yakni *KMP* dan *Boyer-Moore*. Hal utama yang menjadi penyebabnya ialah perbedaan urutan pencocokan *token* yang dilakukan pada gambar.




*Gambar 10 (atas) Ilustrasi masalah implementasi algoritma KMP, (bawah) 2 kemungkinan solusi dari masalah tersebut*

Perhatikan **Gambar 10 (atas)**. Misalkan terjadi *mismatch* pada *token* ke-9 pada *pattern*. Berdasarkan prinsip *KMP*, maka seharusnya *pattern* digeser sebanyak 8 - *b*(8) = 6 *token* ke kanan. Akan tetapi, hal ini tidak dimungkinkan karena urutan pencocokan *token* pada gambar yang berbeda dengan *string* biasa. Pada **Gambar 10 (bawah)**, digambarkan 2 kemungkinan pergeseran yang seharusnya sesuai dengan prinsip *KMP*, yakni (1) bergeser ke bawah dan kanan sampai *pattern* sejajar dengan *longest-prefix-suffix*, atau (2) bergeser ke kanan sejauh 6 *token* seperti pada *string* biasa. Akan tetapi, seperti yang dilihat, keduanya bergeser secara tidak logis dan memungkinkan melewatkan solusi saat bergeser.

Untuk mengatasi masalah ini, penulis mengimplementasikan *border function* hanya pada baris pertama *pattern*. Dengan begitu, pergeseran akan lebih aman. Akan tetapi, hal ini menyebabkan pergeseran yang kurang lebih sama dengan *brute force*. Hanya pada beberapa kasus, akan lebih baik dari *brute force* (kasus uji 3).

4. Masalah format gambar digital

Sejauh ini, program yang dihasilkan hanya dapat memproses satu jenis format gambar, yakni *PNG* (*Portable Network Graphic*). Hal yang menjadi pertimbangan dalam masalah ini adalah kompresi *pixel* yang bisa terjadi pada saat melakukan *cropping* (pemotongan) gambar. Misalkan, kita mempunyai sebuah gambar lukisan Monalisa dan melakukan *cropping*. Pada saat menyimpan gambar hasil *cropping* tersebut, ada kemungkinan potongan gambar tidak memiliki informasi pixel yang sama dengan gambar aslinya. Hal ini bergantung kepada format penyimpanan gambar yang dipilih, seperti *file JPG* yang melakukan pengubahan terhadap nilai *pixel* untuk mengurangi ukuran *file* (kompresi). Sebaliknya, *file PNG* bersifat *non-lossy* yang tetap mempertahankan informasi *pixel* tanpa melakukan kompresi.[6]

## V. Simpulan dan Saran

*Pattern matching* adalah algoritma pemecahan masalah yang banyak diaplikasikan dalam sebuah program komputer. Berbagai pendekatan yang terdapat dalam tulisan ini adalah pendekatan yang sederhana dan kemungkinan tidak banyak digunakan dalam kehidupan nyata. Pencarian pola dalam gambar menggunakan algoritma yang penulis bahas, pun masih terbilang sangat naif dan masih perlu dipertanyakan lagi latar belakang masalah dan solusi yang penulis buat.

Pencarian pola dalam gambar sebenarnya telah banyak dikembangkan oleh perusahaan teknologi raksasa dunia, misalnya *Facebook* dan *Google*. Untuk ke depannya, pencocokan pola dalam gambar sebaiknya tidak menggunakan algoritma *exact match* karena tidak memiliki manfaat banyak untuk kita.

Penulis meminta maaf apabila ada kesalahan dalam penulisan atau penyampaian materi tulisan ini. Semoga pembaca bisa mendapatkan manfaat sebanyak-banyaknya dan, jika berkenan, memberi saran dan kritik kepada penulis.


## Ucapan Terima Kasih

Penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa atas selesainya pembuatan tulisan ini. Tidak lupa, penulis berterima kasih kepada dosen mata kuliah IF2211 Strategi Algoritma, yakni Bapak Rinaldi Munir dan Ibu Nur Ulfa Maulidevi yang telah memberikan materi dan bimbingan, baik di dalam maupun di luar kelas, yang bermanfaat dalam pembuatan tulisan ini.


## Referensi


[1] Munir, Rinaldi. 2009. *Diktat IF2211 Strategi Algoritma*.

[2] *Digital Image File Formats*. American Institute for Conservation - Electronic Media Group.

[3] Diakses dari http://windows.microsoft.com/ pada 6 Mei 2016

[4] Diakses dari https://www.prestashop.com/forums/topic/412536-mixed-content-warnings-in-backoffice-presta-1609/ pada 6 Mei 2016

[5] Diakses dari http://mulpix.com/instagram/일본노모_기모노.html pada 6 Mei 2016

[6] Z, Eris. 2010. *What's the Difference Between JPG, PNG, and GIF?*. diakses dari http://www.howtogeek.com/howto/30941/whats-the-difference-between-jpg-png-and-gif/ pada 6 Mei 2016


## 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, 6 Mei 2016

Muhammad Farhan Majid (13514029)