# Penggunaan Algoritma Pencocokkan Pola pada Sistem *Barcode*

Hishshah Ghassani - 13514056
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, JL. Ganesha 10 Bandung 40132, Indonesia*
*13514056@std.stei.itb.ac.id*

***Abstrak*—Salah satu penerapan algoritma pencocokkan pola adalah pada sistem *barcode*. *Barcode* (dalam bahasa Indonesia berarti kode baris) adalah kode sederhana yang digunakan untuk menyimpan data optik spesifik (seperti kode produksi) yang nantinya akan dibaca oleh mesin. *Barcode* terdiri dari susunan garis hitam-putih dengan ketebalan berbeda-beda yang biasanya merepresentasikan angka atau alfanumerik. Pada makalah ini, akan dibahas mengenai sistem *barcode* dan penerapan algoritma pencocokkan pola, yaitu algoritma *brute force*, Knuth-Morris-Pratt, dan algoritma Boyer-Moore pada penentuan jenis produk melalui pembacaan barcode.**

***Kata kunci—barcode; sistem barcode; database; pencocokan pola; Brute Force; Knuth-Morris-Pratt; Boyer-Moore***

## I. PENDAHULUAN

Zaman sekarang, *barcode* dapat ditemukan di mana-mana. Seperti pada kemasan makanan dan minuman, paspor, soal ujian, buku, tiket pesawat, kartu identitas, obat-obatan, dan produk-produk lainnya. Pada awalnya, *barcode* digunakan untuk memudahkan sistem pemeriksaan barang-barang di swalayan agar otomatis. Namun, penggunaannya sudah meluas sehingga sekarang *barcode* merupakan bagian penting dalam peradaban modern. Terdapat beberapa kategori *barcode* berdasarkan kegunaannya, yaitu untuk keperluan *retail*, untuk keperluan *packaging*, untuk penerbitan, untuk keperluan farmasi, untuk keperluan *non retail*, dan untuk keperluan lainnya.

*Barcode* ditemukan oleh Bernard Silver dan Norman Joseph Woodland pada tahun 1952. Mereka adalah murid lulusan Drexel Institute of Technology di Philadelphia. Awalnya, Drexel Institute of Technology diminta oleh suatu toko makanan untuk membuat sistem pembacaan informasi produk. Solusi pertama yang muncul dari Woodland adalah tinta yang sensitif terhadap sinar ultraviolet. Namun hal tersebut ditolak karena mahal dan tidak stabil. Kemudian, mereka berdua mencari solusi lain dan akhirnya menemukan *barcode*.

*Barcode* terdiri dari garis hitam dan putih dengan ketebalan yang berbeda-beda. *Barcode* dapat dibaca oleh pemindai optik yang biasa disebut *barcode scanner* (pembaca kode baris). *Barcode* diciptakan karena perangkat seperti komputer lebih mudah membaca sesuatu yang digital daripada angka yang bersifat analog. Garis-garis pada *barcode* dibuat dengan kontras tinggi sedemikian sehingga sensor optik CCD (*Charge Couple Device*) atau laser pada *scanner barcode* dapat mengenalinya dengan mudah dan kemudian menerjemahkannya menjadi angka. Kode *barcode* yang dibaca oleh *scanner* tersebut kemudian dicocokkan pada suatu *database* untuk diketahui informasinya.

Untuk mencocokkan kode *barcode* yang dibaca dengan *database*, tentu diperlukan suatu metode dalam pencarian kode *barcode* di *database* tersebut. Banyak cara yang dapat dilakukan, dan menurut penulis salah satu caranya adalah dengan pencocokan pola, mirip dengan pencarian suatu kata dalam file teks. Cara tersebut dapat dilakukan dengan algoritma pencocokan pola.

Pada mata kuliah IF2211 Strategi Algoritma, salah satu materinya adalah tentang beberapa algoritma pencocokkan pola yang sering digunakan, yaitu *brute force*, Knuth-Morris-Pratt, dan Boyer-Moore. Dengan adanya pola yang jelas pada sistem *barcode*, penulis merasa bahwa algoritma pencocokkan pola tersebut dapat digunakan untuk menentukan jenis produk melalui pembacaan *barcode,* yaitu dengan cara pencarian kode produk pada *database*.

## II. TEORI DASAR

### *A. Jenis-Jenis Barcode*

Berikut adalah beberapa jenis *barcode* yang umum digunakan:

*1) Code 39*

Gambar 1 Contoh Barcode Jenis Code 39

Code 39, dikenal juga sebagai code 3 of 9, adalah kode pertama berupa alfanumerik, yaitu huruf besar, angka, dan

karakter tambahan lainnya seperti - $ / + % * dan spasi. Code 39 diawali dan diakhiri dengan * yang merupakan karakter penanda mulai dan berhenti.

*2) Universal Product Code (UPC)*


Gambar 2 Contoh Barcode Jenis UPC

UPC dibuat oleh Amerika Serikat yang mewakili Kode Produk Universal. Kode-kode UPC mudah dilihat mata yang tak terlatih dan hanya mengodekan angka. UPC hanya akan mengkodekan dua belas digit (UPC-A) dan delapan digit (UPC-E), dengan masing-masing 1 digit sebagai *check digit* dan sisa digit lainnya untuk data.

*3) European Articles Numbering (EAN)*


Gambar 3 Contoh Barcode Jenis EAN

Seperti UPC, EAN juga hanya mengodekan angka. Terdapat dua tipe utama pada kode EAN, yaitu EAN-13 dan EAN-8. EAN-13 terdiri dari 13 digit angka yang dibagi menjadi 4 kelompok: 3 angka pertama mewakili negara (kodenya diatur oleh EAN International), 4 angka berikutnya untuk perusahaan pengguna, 5 angka berikutnya untuk kode produk pada tiap perusahaan, dan 1 angka terakhir sebagai *check digit*. EAN-8 mirip dengan EAN-13, hanya saja terdiri dari 8 digit dengan pembagian: 3 angka pertama mewakili negara, 2 digit berikutnya sebagai nomor perusahaan, dua digit berikutnya untuk kode produk, dan 1 digit terakhir sebagai *check digit*.

Tipe barcode yang sering digunakan di Indonesia adalah EAN-13, dengan 3 kode awal yang merupakan kode negara adalah 899. EAN-13 juga dijadikan standar *barcode retail* di seluruh dunia (kecuali Amerika dan Kanada).

## B. Sistem Barcode

Pada *barcode*, sebenarnya terdapat beberapa kolom dengan besar yang sama. Pada saat *barcode* dibaca, yang dilakukan sebenarnya adalah melihat apakah pada kolom tersebut warna hitam atau putih. Warna hitam pada *barcode* merepresentasikan angka 1 sedangkan warna putih merepresentasikan angka 0. Hal ini cocok dengan konsep digital karena hanya ada dua sinyal data yang dapat dikenali pada konsep digital dan sinyal data tersebut bersifat boolean, yaitu angka 1 dan 0. Kemudian komputer memulai membaca *barcode* dari kiri ke kanan dan mengisi kolom-kolom tersebut dengan angka 1 dan 0, sehingga membentuk angka-angka yang terdiri dari 1 dan 0. Angka-angka tersebut terbagi menjadi beberapa bagian, yaitu 3 bagian sebagai penanda di kiri, tengah, dan kanan, serta bagian lainnya menandakan angka-angka yang biasanya berada di bawah *barcode* (misalnya pada jenis UPC dan EAN). Penanda kiri, tengah, dan kanan tersebut fungsinya adalah untuk memberi tanda pada komputer apakah *barcode* dibaca dari kanan atau kiri. Jadi, tidak masalah apabila pembacaan barcode terbalik atas dan bawahnya karena adanya penanda tersebut.

Pembacaan *barcode* dapat juga dilakukan dengan melihat ketebalan garisnya. Masing-masing garis pada *barcode* memiliki ketebalan yang berbeda-beda. Ketebalan itulah yang menerjemahkan suatu nilai. Garis paling tipis adalah "1", kemudian yang sedang adalah "2", yang lebih tebal adalah "3", dan yang paling tebal adalah "4". Setiap digit angka 0-9 terdiri dari empat urutan angka yang dijelaskan pada tabel berikut:

Tabel 1 Urutan Angka Penyusun Angka 0-9

| | |
|---|---|
| 0 | 3211 |
| 1 | 2221 |
| 2 | 2122 |
| 3 | 1411 |
| 4 | 1132 |
| 5 | 1231 |
| 6 | 1114 |
| 7 | 1312 |
| 8 | 1213 |
| 9 | 3112 |

Setelah *barcode* dibaca, kemudian angka-angka yang dihasilkan dicocokkan dengan *database* yang ada. *Database* tersebut menyimpan informasi-informasi produk yang dapat digunakan. *Database* juga berfungsi untuk mempermudah dalam memperbarui barang-barang yang memiliki *barcode*. Misalnya untuk memperbarui harga atau jumlah suatu barang pada suatu toko, maka pengelola dapat mengubah *database*nya saja tanpa harus membuat *barcode* baru.

## C. Algoritma Pencocokan Pola

Algoritma pencocokkan pola adalah algoritma yang digunakan untuk mencari pola tertentu pada suatu teks, apakah terdapat pola tersebut atau tidak. Algoritma ini juga dapat menemukan posisi pola tersebut di dalam teks (jika ada).

Dalam dunia nyata, algoritma ini sangat sering digunakan. Misalnya untuk pencarian dalam editor teks, *web search engine* (seperti Google), analisis citra, pencocokkan rantai asam amino pada rantai DNA, dan sebagainya.

Algoritma pencocokkan pola yang umum diguanakan adalah algoritma *brute force*, Knuth-Morris-Pratt (KMP), dan Boyer-Moore. Namun, algoritma *brute force* memiliki performansi yang sangat buruk sehingga jarang digunakan. Berikut penjelasan masing-masing algoritma tersebut:

*1) Algoritma Brute Force*

Algoritma *brute force* mencari kecocokan antara pola dan teks satu per satu dari kiri ke kanan. Apabila ditemukan ketidakcocokan, pola akan bergeser satu karakter ke kanan

dan mulai mencocokkan lagi dari awal. Algoritma ini akan bergeser terus setiap ditemukan ketidakcocokan, sampai pola berada diujung teks.

Apabila teks berada dalam array T[1..n] dan pola berada dalam array P[1..n], maka algoritma *brute force* untuk pencocokan pola adalah sebagai berikut:

- P dicocokkan pada awal T
- Bandingkan setiap karakter pada P dengan karakter dalam T dari kiri ke kanan, hingga semua karakter di P cocok dengan T (pencarian berhasil) atau dicumpai ketidakcocokan karakter antara P dan T (pencarian belum berhasil)
- Bila terdapat ketidakcocokan antara P dan T, dan T belum habis, maka geser P satu karakter ke kanan dan ulangi langkah pada poin kedua.

Berikut adalah contoh penggunaan algoritma *brute force* pada pencocokan pola:

Pola: NOT

Teks: NOBODY NOTICED HIM

```
  NOBODY NOTICED HIM
1 NOT
2  NOT
3   NOT
4    NOT
5     NOT
6      NOT
7       NOT
8        NOT
```

Kesimpulan: pola NOT ditemukan pada posisi indeks ke-8 dari awal teks (T[8]).

Berikut adalah *pseudocode* algoritma *brute force* pada pencocokan pola:

```
procedure BruteForce(input m, n: integer,
                     input P: array[1..m] of char,
                     input T: array[1..n] of char,
                     output idx: integer)
{ Mencari kecocokan pola P di dalam teks T. Jika
  ditemukan P di dalam T, lokasi awal kecocokan
  disimpan ke dalam variabel idx. }

Deklarasi
  s, j: integer
  found: boolean

Algoritma
  s ← 0
  found ← false
  while (s ≤ m-n) and (not found) do
    j ← 1
    while (j ≤ m) and (P[j] = T[s+j]) do
      j ← j + 1
    endwhile
    { j > m atau P[j] ≠ T[s+j] }
    if (j = m) then { pencarian berhasil }
      ketemu ← true
    else { pencarian belum berhasil }
      s ← s + 1 { geser satu karakter ke kanan}
    endif
  endwhile
  { s > n-m atau found }

  if ketemu then
    idx ← s + 1
  else
    idx ← -1
endif
```

Kompleksitas algoritma *brute force* untuk pencocokan string dapat dilihat dari jumlah perbandingan yang dilakukan. Untuk kasus terbaik, yaitu jika karakter pertama P tidak sama dengan karakter pada T kecuali pada pencocokan terakhir, kompleksitas algoritmanya adalah O(n). Sedangkan untuk kasus terburuk, yaitu ketika semua karakter pada P kecuali karakter terakhir sama dengan karakter pada T, komplesitas algoritmanya adalah O(mn).

*2) Algoritma Knuth-Morris-Pratt (KMP)*

Algoritma ini dikambangkan oleh D. E. Knuth, bersama dengan J. H. Morris dan V. R. Pratt. Algoritma KMP mirip dengan algoritma *brute force*, namun algoritma ini bergeser dengan lebih "pintar". Algoritma ini menggunakan fungsi pinggiran (*border function*) yang berguna untuk menentukan seberapa banyak pola digeser ketika ditemukan ketidakcocokan antara teks dan pola. Sehingga pada algoritma KMP, pergeseran tidak selalu hanya satu karakter saja seperti pada algoritma *brute force*. Oleh karena itu, waktu pencarian pada algoritma KMP berkurang secara signifikan dibandingkan algoritma *brute force*.

Proses awal pada algoritma KMP adalah menentukan fungsi piggiran, yang menandakan pergeseran sebesar mungkin yang dapat dilakukan. Fungsi pinggiran adalah ukuran terbesar dari *prefix* yang juga merupakan *suffix*. Fungsi pinggiran ini dapat mencegah pergeseran dan perbandingan yang tidak berguna seperti pada algoritma *brute force*. Fungsi pinggiran hanya bergantung dengan karakter-karakter yang ada pada pola, bukan karakter pada teks. Oleh karena itu, fungsi pinggiran dapat ditentukan sebelum pencarian dilakukan.

Sebagai contoh, apabila diberikan pola P = abaaba, maka fungsi pinggirannya adalah sebagai berikut:

Tabel 2 Fungsi Pinggiran untuk P = abaaba

| j | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| P[j] | a | b | a | a | b | a |
| b(j) | 0 | 0 | 1 | 1 | 2 | 3 |

Setelah didapatkan fungsi pinggiran, lakukan pencocokan antara pola P dengan teks T. Apabila terdapat karakter yang tidak sama antara P dan T, lakukan pergeseran sebanyak panjang karakter awal yang bersesuaian (l) dikurangi fungsi pinggiran karakter P terakhir yang masih cocok (b(l)). Misalnya ditemukan ketidak cocokan pada karakter ke-6, maka dengan fungsi

pinggiran di atas, didapatkan l = 5 dan b(5) = 2, sehingga pergeseran dilakukan sebanyak l – b = 5 – 2 = 3. Jadi, pergeseran dilakukan sebanyak 3 karakter dan perbandingan dimulai lagi pada posisi ketiga dari awal pola.

Berikut adalah *pseudocode* algoritma KMP:

```
procedure KMP(input m, n: integer,
              input P: array[1..m] of char,
              input T: array[1..n] of char,
              output idx: integer)
{ Mencari kecocokan pola P di dalam teks T. Jika
  ditemukan P di dalam T, lokasi awal kecocokan
  disimpan ke dalam variabel idx. }

Deklarasi
  i, j: integer
  found: boolean
  b : array[1..m] of integer

Algoritma
  HitungPinggiran(m, P, b) { Menghitung nilai fungsi
  pinggiran b[1..m] untuk pola P[1..m] }
  i ← 1
  j ← 0
  found ← false
  while (i ≤ n) and (not found) do
    while (j > 0) and (P[j+1] ≠ T[i]) do
      j ← b[j]
    endwhile

    if (P[j+1] = T[i]) then
      j ← j + 1
    endif
    if (j = m) then
      found ← true
    else
      i ← i + 1
    endif
  endwhile

  if ketemu then
    idx ← i – m + 1
  else
    idx ← -1
endif
```

Kompleksitas algoritma untuk menghitung fungsi pinggiran adalah O(m). Sedangkan kompleksitas algoritma untuk pencarian string adalah O(n). Sehingga kompleksitas algoritma KMP adalah O(m+n).

*3) Algoritma Boyer-Moore*

Algoritma ini dikembangkan oleh Robert S. Boyer dan J. Strother Moore. Berbeda dengan algoritma *brute force* dan algoritma KMP yang melakuakn perbandingan dari kiri pola, algoritma Boyer-Moore melakukan perbandingan dari kanan pola sehingga lebih banyak informasi yang didapat.

Pada algoritma Boyer-Moore, dikenal istilah *last occurrence*, yaitu posisi suatu karakter terakhir ditemukan pada suatu pola. Misalkan diberi pola P = abaaba. Maka *last occurrence* dari a (L(a)) adalah 6 dan *last occurrence* dari b (L(b)) adalah 5.

Apabila saat pencocokan terdapat ketidakcocokan antara pola P dan teks T (misalkan karakter x pada T), maka akan dilakukan pergeseran berdasarkan kasusnya. Berikut tiga kasus serta pergeseran pada algoritma Boyer-Moore:

a. Kasus pertama adalah jika x terdapat pada P dan *last occurrence* x lebih kecil dari j (dengan j adalah posisi karakter yang sedang diperiksa pada P), maka lakukan pergeseran ke kanan sedemikian sehingga karakter x pada P bersesuaian dengan pada T.

b. Kasus kedua adalah jika x terdapat pada P namun *last occurrence* x lebih besar dari j, maka lakukan pergeseran ke kanan sebanyak satu karakter.

c. Kasus ketiga adalah jika x tidak terdapat pada P, maka lakukan pergeseran ke kanan sedemikian sehingga karakter pertama pada P bersesuaian dengan posisi karakter x + 1 pada T.

Berikut adalah contoh penggunaan algoritma Boyer-Moore:

Pola: NOT

Teks: NOBODY NOTICED HIM

```
  NOBODY NOTICED HIM
1 NOT
2    NOT
3       NOT
4         NOT
```

Berikut adalah *pseudocode* algoritma Boyer-Moore:

```
procedure BoyerMoore(input m, n: integer,
                     input P: array[1..m] of char,
                     input T: array[1..n] of char,
                     output idx: integer)
{ Mencari kecocokan pola P di dalam teks T. Jika
  ditemukan P di dalam T, lokasi awal kecocokan
  disimpan ke dalam variabel idx. }

Deklarasi
  i, j: integer
  found: boolean
  L : array[1..m] of integer

Algoritma
  LastOccurrence(m, P, L) { Menghitung nilai last
  occurrence L[1..m] untuk pola P[1..m] }
  i ← m
  j ← m
  found ← false
  repeat
    if (P[j] = T[j]) then
      if (j = 0) then
        found ← true
      else
        i ← i – 1
        j ← j – 1
    else
      i ← i + m – min(j, 1 + L[T[i]])
      j ← m – 1
```

```
  until (i ≤ n)

  if ketemu then
    idx ← i
  else
    idx ← -1
endif
```

Algoritma Boyer-Moore lebih cepat daripada algoritma KMP apabila jumlah karakter banyak, namun tidak efektif apabila jumlah karakter sedikit. Kompleksitas algoritma Boyer-Moore untuk kasus terburuk adalah O(nm + A).

## III. Analisis Penggunaan Algoritma Pencocokan Pola pada Sistem *Barcode*

Seperti yang dijelaskan pada bab sebelumnya, terdapat suatu *database* pada sistem *barcode* yang berisi informasi mengenai produk yang memiliki *barcode*. Kemudian, untuk mengetahui informasi produk tersebut, dilakukan pembacaan *barcode* oleh *scanner* dan kemudian dapat digunakan algoritma pencocokan pola untuk mencari produk tersebut di dalam *database*.

Misalnya di suatu toko, terdapat benda A, B, dan C dengan masing-masing kode produk adalah 8997977889788, 8995077600135, dan 8993175538572. Kemudian pada toko tersebut terdapat database sebagai berikut:

Tabel 3 Contoh Database pada Sebuah Toko

| Kode Barang | Nama Barang | Jumlah Stok | Harga Jual |
|---|---|---|---|
| 8997977889788 | A | 14 | Rp6.000,00 |
| 8995077600135 | B | 5 | Rp15.000,00 |
| 8993175538572 | C | 21 | Rp4.000,00 |

Kemudian, *scanner* mengirimkan hasil pembacaan suatu *barcode* bernilai 8995077600135 (jumlah digit = 13), dan ingin diketahui nama barang dari hasil *barcode* tersebut serta harga jualnya.

Di sinilah peran algoritma pencocokan pola, yaitu mencari kode barang tersebut pada *database* agar diketahui nama dan harga jualnya. Pola yang dicari adalah hasil pembacaan *barcode*, dan teksnya adalah semua kode barang pada *database* dan dipisahkan dengan spasi. Misalkan indeks pola ditemukan pada teks adalah idx, dan jumlah digit *barcode* (panjang pola) adalah L. Untuk mengetahui barang keberapa yang sedang dicari, dapat menggunakan rumus yaitu: idx modulo L. Rumus tersebut dapat digunakan apabila jumlah digit semua kode pada *database* tersebut sama. Karena jumlah digit *barcode* pada masing-masing toko biasanya sama, maka rumus tersebut dapat digunakan.

Dalam makalah ini, akan dicoba pendekatan dengan tiga buah algoritma pencocokan pola, yaitu algoritma *brute force*, algoritma KMP, dan algoritma Boyer-Moore, untuk menemukan kode barang pada *database*.

*1) Algoritma Brute Force*

```
Pola: 8995077600135

Teks: 8997977889788 8995077600135 8993175538572
```

```
  8997977889788 8995077600135 8993175538572
1 8995077600135
2  8995077600135
3   8995077600135
4    8995077600135
5     8995077600135
6      8995077600135
7       8995077600135
8        8995077600135
9         8995077600135
10         8995077600135
11          8995077600135
12           8995077600135
13            8995077600135
14             8995077600135
15              8995077600135
```

Pola ditemukan pada iterasi ke-15 dan pada indeks ke-15 (idx = 15). Urutan barang yang dicari = idx mod L = 15 mod 13 = 2. Maka, barang yang dicari adalah barang ke-2.

*2) Algoritma Knuth-Morris-Pratt*

Fungsi pinggiran:

Tabel 4 Fungsi Pinggiran untuk P = 8995077600135

| j | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P[j] | 8 | 9 | 9 | 5 | 0 | 7 | 7 | 6 | 0 | 0 | 1 | 3 | 5 |
| b(j) | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Pencarian:

```
Pola: 8995077600135

Teks: 8997977889788 8995077600135 8993175538572

  8997977889788 8995077600135 8993175538572
1 8995077600135
2  8995077600135
3   8995077600135
4    8995077600135
5     8995077600135
6      8995077600135
7       8995077600135
8        8995077600135
9         8995077600135
10         8995077600135
11          8995077600135
12           8995077600135
13            8995077600135
14             8995077600135
15              8995077600135
```

Pola ditemukan pada iterasi ke-15 dan pada indeks ke-15 (idx = 15). Urutan barang yang dicari = idx mod L = 15 mod 13 = 2. Maka, barang yang dicari adalah barang ke-2.

*3) Algoritma Boyer-Moore*

*Last occurrence*:

Tabel 5 *Last Occurrence* untuk P = 8995077600135

| x | 8 | 9 | 5 | 0 | 7 | 6 | 1 | 3 |
|---|---|---|---|---|---|---|---|---|
| L(x) | 1 | 3 | 13 | 10 | 7 | 8 | 11 | 12 |

Pencarian:

```
Pola: 8995077600135

Teks: 8997977889788 8995077600135 8993175538572

  8997977889788 8995077600135 8993175538572
1 8995077600135
2               8995077600135
3                 8995077600135
```

Pola ditemukan pada iterasi ke-3 dan pada indeks ke-15 (idx = 15). Urutan barang yang dicari = idx mod L = 15 mod 13 = 2. Sehingga barang yang dicari adalah barang ke-2.

Dari hasil ketiga algoritma tersebut, didapat bahwa kode barang 8995077600135 ditemukan pada barang kedua, yang berarti nama barang tersebut adalah B dan harga jualnya senilai Rp15.000,00. Dapat dilihat bahwa algoritma *brute force* dan KMP menemukan pola pada iterasi ke-15, sedangkan algoritma Boyer-Moore langsung menemukannya pada iterasi ke-3.

## IV. Kesimpulan

Algoritma pencocokan pola dapat digunakan dalam sistem *barcode* untuk mengetahui informasi produk hasil pembacaan *barcode*. Pertama, *scanner* membaca *barcode* dan mengirimkan hasilnya untuk dicari pada *database*. Kemudian algoritma tersebut mencari kode barang hasil pembacaan *barcode* pada *database* dengan cara yang sudah dibahas. Setelah didapatkan kode yang tepat, dicari barang keberapa barang tersebut pada database, dengan rumus idx modulo L (di mana idx adalah indeks pola ditemukan pada teks dan L adalah jumlah digit *barcode*), lalu dilihat informasi lainnya yang ingin diketahui. Dapat dilihat bahwa pada kasus ini, algoritma Boyer-Moore jauh lebih efisien daripada algoritma *brute force* dan algoritma KMP. Hal ini disebabkan jumlah karakternya banyak.


## Ucapan Terima Kasih

Pada bagian ini, penulis ingin bersyukur kepada Allah SWT atas segala rahmat dan berkahnya sehingga penulis dapat menyelesaikan makalah ini tepat waktu. Terima kasih juga penulis sampaikan kepada orangtua dan keluarga, yang selalu mendukung, mendoakan, dan memberi semangat kepada penulis dalam menjalani kuliah. Kemudian, ucapan terima kasih juga penulis sampaikan kepada Bapak Rinaldi Munir dan Ibu Nur Ulfa Maulidevi, selaku dosen mata kuliah IF2211 Strategi Algoritma, atas bimbingan, dukungan, dan referensi-referensi yang membantu dalam penyelesaian makalah ini. Terakhir, terima kasih kepada rekan-rekan dan pihak lain yang membantu dalam proses pembuatan makalah ini.


## Daftar Referensi


[1] http://dibukasaja.blogspot.co.id/2013/04/mengenal-apa-itu-barcode.html, diakses pada 1 Mei 2016 pukul 11:00 WIB.

[2] Munir, Rinaldi. 2009. *Diktat Kuliah IF2211 Strategi Algoritma*. Bandung: Program Studi Teknik Informatika.

[3] http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/StringMatch/boyerMoore.htm, diakses pada 1 Mei 2016 pukul 14:14 WIB.

[4] http://www.handalsoftware.com/component/content/article/51-spotlight-news-1/115-kegunaan-barcode-dan-jenis-jenis-barcode.html, diakses pada 1 Mei 2016 pukul 16:34 WIB.

[5] http://www.kiosbarcode.com/blog/komponen-sistem-kerja-barcode/, diakses pada 5 Mei 2016 pukul 17:38 WIB.

[6] http://www.winnsolutions.com/how-does-a-barcode-work/, diakses pada 5 Mei 2016 pukul 21:23 WIB.


## 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

Hishshah Ghassani - 13514056