Penerapan Pencocokan String dalam Pemindaian Barcode Barang di Supermarket

Nabelanita Utami 13519104
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519104@std.stei.itb.ac.id

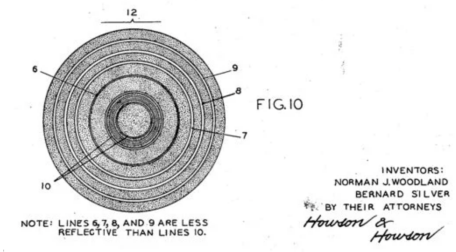
***Abstract*—Kegiatan berbelanja merupakan suatu hal yang tidak pernah lepas dari kehidupan sehari-hari, terutama berbelanja kebutuhan rumah tangga. Dalam berbelanja baik kebutuhan rumah tangga ataupun membeli barang lain seperti pakaian, kita hampir selalu bertemu dengan *barcode*, yaitu sebuah data optik yang tampak tidak bermakna, tapi sangat krusial dalam menentukan identitas akan barang yang dibeli. Dalam barcode tersebut, tersimpan kode-kode yang ternyata memiliki makna penting. Dalam pemindaian barcode dengan menggunakan alat *laser scanner*, akan terbentuk sekumpulan angka yang menyimpan data mulai dari negara asal hingga kode unik barang tersebut. Angka-angka tersebut dapat kita kenali dengan menggunakan salah satu teknik pencocokan pola yaitu dengan menggunakan regular expression.**

Keywords—barcode; scan; pencocokan pola; string matching;

I. PENDAHULUAN

Dalam berbelanja, barang yang dibeli umumnya akan dipindai dengan sebuah alat ketika kita hendak membayar. Setelah dipindai, harga serta informasi lain mengenai barang tersebut akan ditampilkan di layar. Informasi-informasi tersebut berasal dari barcode, yaitu sekumpulan kode berbentuk garis yang masing-masing memiliki ketebalan berbeda sesuai data yang direpresentasikannya.

Penggunaan barcode dimulai pada tahun 1948 oleh dua mahasiswa bernama Norman J Woodland dan Bernard Silver, yang memiliki gagasan untuk mengubah metode pembayaran pelanggan di supermarket agar menjadi lebih efisien. Desain awal barcode pada awalnya jauh berbeda dengan yang digunakan saat ini, yaitu berbentuk lingkaran dengan lingkaran-lingkaran lebih kecil didalamnya.



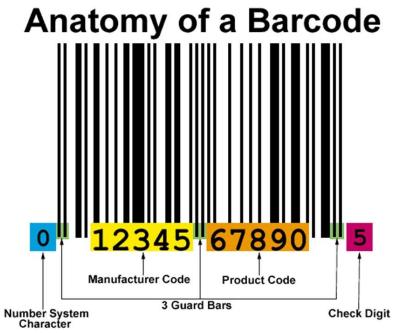
Gambar 1.1 Desain awal barcode ketika pertama kali diciptakan

Sumber: trackabout.com

Barcode jenis linear seperti yang digunakan saat ini baru ditemukan pada tahun 1960. Mesin pemindai untuk barcode jenis linear pada saat itu juga tidak seperti sekarang. Saat awal diciptakan, mesin pemindai barcode berukuran hingga sebesar mesin pendingin. Karena dulunya barcode tersebut diciptakan untuk perusahaan kereta api, mesin pemindai akan menyala ketika terdapat kereta api yang melintas serta akan mengeluarkan cahaya dengan daya 500 watt.

Mesin pemindai tersebut tentu saja bukan yang paling efisien karena cahaya 500 watt sangatlah boros. Oleh karena itu, diciptakanlah sebuah mesin pemindai berbasis laser pada tahun 1970-an yang jauh lebih efisien. Mesin pemindai laser tersebut awalnya digunakan oleh sebuah perusahaan mobil. Barcode linear beserta alat pemindai laser ini baru digunakan di supermarket pada tahun 1980 dan tetap digunakan di banyak sekali bidang hingga saat ini.

Barcode umumnya memiliki sekumpulan kode dibawahnya. Secara singkat, garis-garis tersebut jika dipindai akan menghasilkan kode yang sama dengan yang tertulis. Namun, memasukkan secara manual kode-kode tersebut akan memakan waktu yang lama, apalagi jika pelanggan membeli barang yang banyak. Oleh karena itu, dibentuklah sebuah data optik berupa garis-garis sehingga cukup dilakukan pemindaian saja. Sebuah barcode dapat dibagi menjadi beberapa bagian yang merepresentasikan informasi berbeda seperti yang dapat dilihat di gambar berikut ini.



Gambar 1.1 Diagram anatomi dari sebuah barcode

Sumber: Kiswara.co.id

Dari diagram tersebut, dapat disimpulkan bahwa barcode dari sebuah barang mengikuti pola tertentu. Hal ini berarti bahwa kode dari barcode dapat dengan mudah kita pecahkan dengan teknik pencocokan pola. Dari hasil tersebut, nantinya dapat diketahui informasi secara mendetail mengenai barang yang dipindai. Pada makalah ini, teknik pencocokan pola yang akan digunakan adalah regular expression.

II. Teori Dasar

A. *Pencocokan pola*

Pencocokan pola dalam dunia Informatika merupakan suatu teknik untuk mencari pola tertentu di suatu medium, dalam kasus ini merupakan pencocokan pola berupa string dalam suatu teks yaitu kode barang dari barcode. Tak hanya string, pencocokan pola juga bisa dilakukan dalam pengolahan citra.

Dalam pencocokan pola pada string, terdapat dua elemen yang penting, yaitu T atau teks dan P atau pola. Teks T merupakan string panjang dengan panjang n, sedangkan pola P merupakan string dengan ukuran m, di mana m < n. Dalam string ada juga istilah yang dipakai dalam pencocokan pola, yaitu prefiks dan sufiks. Prefiks atau *prefix* merupakan substring dari string S[0..k], sedangkan sufiks atau *suffix* merupakan substring dari string S[k...m-1], dengan k merupakan indeks antara 0 hingga m-1. Ingat kembali bahwa m merupakan panjang dari string. Misal terdapat string “world”, maka prefiks yang mungkin adalah “w”, “wo” “wor”, ‘worl”, dan “world”, sedangkan sufiks yang mungkin adalah “d”, “ld”, “rld”, “orld”, dan “world”. Aplikasi pencocokan pola pada string ini sering dipakai pada pencarian teks di *text editor* atau mesin pencari seperti Google.

Terdapat beberapa teknik yang digunakan dalam pencocokan pola, yaitu dengan algoritma Brute Force, algoritma Knuth-Morris Pratt, algoritma Boyer-Moore, dan regular expression.

B. *Algoritma Brute Force*

Algoritma Brute Force atau seringkali disebut dengan algoritma naif merupakan teknik pencocokan pola dengan cara “tradisional”, yaitu dengan mencocokan tiap string satu per satu. Teknik ini merupakan cara mencocokan string yang paling tidak mangkus diantara yang lainnya.

Algoritma brute force mencocokkan string pola dimulai dari indeks pertama string teks. Apabila terjadi ketidakcocokan atau *mismatch*, pencocokan pola akan diulangi dari indeks selanjutnya, dalam kasus ini adalah indeks kedua dari string teks. Hal ini menyebabkan algoritma brute force tampak seperti “mundur”, karena jika apabila *mismatch* terjadi di indeks terakhir pola, pencocokan string harus dilakukan kembali dari awal, padahal sebelumnya telah dilakukan. Berikut ini merupakan gambaran dari pencocokan string dengan algoritma brute force.

Teks: NOBODY NOTICED HIM
Pattern: NOT

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

Gambar 2.1 Contoh pencocokan pola string dengan menggunakan algoritma Brute Force

Sumber: Pencocokan String (String/Pattern Matching), Rinaldi Munir

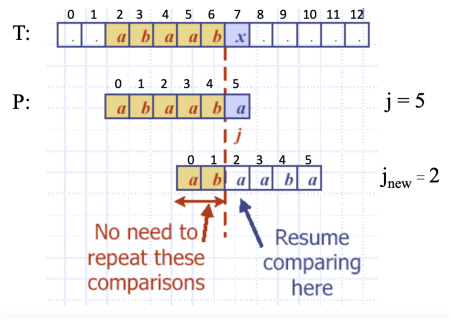
Jumlah perbandingan dengan algoritma Brute Force, yaitu apabila teks tidak pernah sama dengan pola kecuali pada bagian akhir, adalah maksimal n kali dengan kompleksitas waktu O(n). Di sisi lain, jumlah perbandingan untuk skenario terburuk, yaitu jika seluruh teks sama dengan pola kecuali pada bagian akhir, adalah m(n–m+1) kali dengan kompleksitas waktu O(mn). Semenara itu, kompleksitas waktu untuk skenario rata-rata adalah O(m+n). Algoritma Brute Force memiliki *pseudocode* sebagai berikut.

```
function BruteForce (input P: string, T: string) → boolean
{ Melakukan string matching secara brute force.  }
{ Mengembalikan true jika terdapat pola pada     }
{ string.                                        }
KAMUS
  n, m: int
  function Length(input s: string) → int
  { Mengembalikan panjang sebuah string }
ALGORITMA
  n ← Length(T)
  m ← Length(P)
  i traversal (0, n-m)
    j ← 0
    while (j < m and T[i+j] = P[j]) do
       j ← j + 1
    if (j = m) then
       → True
    → False
```

C. *Algoritma Knuth-Morris Pratt*

Algoritma Knuth-Morris Pratt atau algoritma KMP melakukan pencocokan string dari kiri ke kanan, mirip dengan algoritma Brute Force, namun melakukan pergeseran dengan cara yang lebih pintar. Ketika terjadi *mismatch*, algoritma

KMP akan melakukan pergeseran dengan besaran tertentu sehingga tidak akan terjadi pencocokan ulang yang "mundur" seperti pada algoritma Brute Force. Pergeseran ini didasari dengan panjang prefiks terbesar pola yang juga merupakan sufiks.



Gambar 2.2 Ilustrasi pencocokan string dengan menggunakan algoritma KMP

Sumber: Pencocokan String (String/Pattern Matching), Rinaldi Munir

Algoritma Knuth-Morris Pratt atau algoritma KMP melakukan pencocokan string dari kiri ke kanan, mirip dengan algoritma Brute Force, namun melakukan pergeseran dengan cara yang lebih pintar. Ketika terjadi *mismatch*, algoritma KMP akan melakukan pergeseran dengan besaran tertentu sehingga tidak akan terjadi pencocokan ulang yang "mundur" seperti pada algoritma Brute Force. Pergeseran ini didasari dengan panjang prefiks terbesar pola yang juga merupakan sufiks.

Indeks baru dari pergeseran diproses dalam sebuah fungsi yang disebut dengan *border function* atau fungsi pinggiran. Berikut ini merupakan contoh dari hasil fungsi pinggiran dari pola "abaaba".

(k = j-1)

j	0	1	2	3	4	5
P[j]	*a*	*b*	*a*	*a*	*b*	*a*
k	-	0	1	2	3	4
b(k)	-	0	0	1	1	2

b(k) is the size of the largest border.

Gambar 2.3 Tabel berisi hasil fungsi pinggiran

Sumber: Pencocokan String (String/Pattern Matching), Rinaldi Munir

Nilai pada baris b(k) merupakan indeks pergeseran apabila terjadi *mismatch* pada pencocokan.

Algoritma KMP memiliki kompleksitas O(m+n), jauh lebih cepat dari algoritma Brute Force. Berikut ini merupakan *pseudocode* dari algoritma KMP.

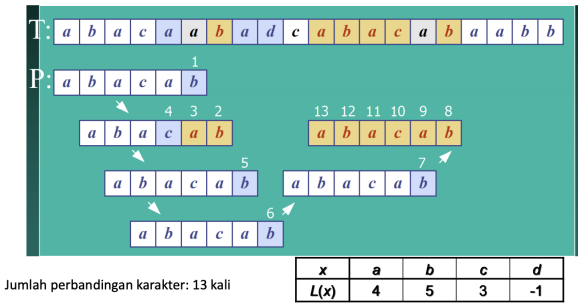
```
function KMP (input P: string, T: string) →
boolean
{ Melakukan string matching secara brute force.  }
{ Mengembalikan true jika terdapat pola pada     }
{ string.                                        }
KAMUS
  n, m, i, j: int
  fail: array of int
ALGORITMA
  n ← Length(T)
  m ← Length(P)
  fail ← BorderFunc(P)

  i ← 0
  j ← 0

  while (i < n) do
    if (P[j] = T[i]) then
      if (j = m -1) then
          → True
      i ← i + 1
      j ← j + 1
    else if (j > 0) then
      j ← fail[j-1]
    else
      i ← i + 1

  → False
```

D. Algoritma Boyer-Moore

Algoritma Boyer-Moore didasari oleh dua teknik, yaitu teknik *looking-glass* dan teknik *character-jump*. Teknik *looking-glass* merupakan teknik mencocokkan pola dengan teks dari kanan ke kiri, yaitu dimulai dari bagian akhir pola. Sementara itu, teknik *character-jump* merupakan "lompatan" yang akan dilakukan apabila terjadi suatu *mismatch*.

Terdapat tiga kasus yang dapat terjadi apabila digunakan algoritma Boyer-Moore. Pertama, jika terdapat suatu karakter pada teks yang juga ada pada pola, maka dilakukan pergeseran hingga terjadi kecocokan. Kedua, jika terdapat suatu karakter pada teks yang juga ada pada pola tapi tidak dimungkinkan dilakukan pergeseran, maka pola akan bergeser maju sejauh satu karakter. Ketiga, jika kedua kasus diatas tidak memenuhi, dilakukan pergeseran hingga teks T[i+1] sejajar dengan pola P[0].

Algoritma Boyer-Moore memerlukan sebuah informasi berupa *last occurence* atau indeks kemunculan terakhir suatu karakter pada pola. Indeks tersebut akan digunakan sebagai dasar dari "lompatan" pola ketika terjadi *mismatch*. Berikut ini merupakan ilustrasi pencocokan string menggunakan algoritma KMP beserta tabel *last occurence*.



x	a	b	c	d
L(x)	4	5	3	-1

Gambar 2.4 Ilustrasi pencocokan string dengan algoritma Boyer-Moore

Sumber: Pencocokan String (String/Pattern Matching), Rinaldi Munir

Algoritma Boyer-Moore memiliki kompleksitas waktu skenario terburuk sebesar O(m+n). Algoritma ini jauh lebih mangkus dari algoritma Brute Force untuk string yang berarti, tetapi tidak bagus untuk string dengan karakter kecil seperti string biner. Berikut ini merupakan *pseudocode* dari algoritma Boyer-Moore.

```
function BoyerMoore (input P: string, T: string) → boolean
{ Melakukan string matching secara brute force.  }
{ Mengembalikan true jika terdapat pola pada     }
{ string.                                        }
KAMUS
  last: array of int
  n, m, i, j: int
  function Length(input s: string) → int
  { Mengembalikan panjang dari suatu string }
  function Unicode(input c: char) → int
  { Mengembalikan bilangan unicode dari suatu    }
  { karakter.                                    }
  function LastOccurrence(input p: string) → array of int
  { Mengembalikan array indeks last occurrence
}
  { dari pattern.                                }
ALGORITMA
  last ← LastOccurrence(P)
  n ← Length(T)
  m ← Length(P)
  i ← m-1

  if (i > n-1) then
    → False

  j ← m-1

  do
    if (P[j] = T[i] then
      if (j = 0) then
        → True
      else
        i ← i-1
        j ← j-1
    else
      i ← i = m - Min(j, 1+last[Unicode(T[i])])
      j ← m-1
  while (i ≤ n-1)

  → False
```

E. Regular Expression

Regular expression atau disingkat menjadi regex merupakan sekumpulan karakter yang digunakan untuk pencarian pola dalam suatu teks. Regex memiliki *syntax* tertentu dalam menspesifikasikan sebuah pola. Daftar *syntax* yang dapat digunakan dalam regex dapat dilihat pada gambar berikut ini.

.	Any character except newline.
\.	A period (and so on for *, \(, \\, etc.)
^	The start of the string.
$	The end of the string.
\d,\w,\s	A digit, word character [A-Za-z0-9_], or whitespace.
\D,\W,\S	Anything except a digit, word character, or whitespace.
[abc]	Character a, b, or c.
[a-z]	a through z.
[^abc]	Any character except a, b, or c.
aa|bb	Either aa or bb.
?	Zero or one of the preceding element.
*	Zero or more of the preceding element.
+	One or more of the preceding element.
{n}	Exactly *n* of the preceding element.
{n,}	*n* or more of the preceding element.
{m,n}	Between *m* and *n* of the preceding element.
??,*?,+?, {n}?, etc.	Same as above, but as few as possible.
(expr)	Capture *expr* for use with \1, etc.
(?:expr)	Non-capturing group.
(?=expr)	Followed by *expr*.
(?!expr)	Not followed by *expr*.

Near-complete reference

Gambar 2.5 Tabel sintaks regular expression

Contoh dari regex adalah "[bcr]at", di mana string "bat", "cat", dan "rat" akan dianggap sebagai *match* dengan regex tersebut.

III. Pembahasan

A. Pemilihan pola yang akan dicari

Mari kita lihat lagi secara sekilas pola barcode yang digunakan di Indonesia pada umumnya. Barang di Indonesia sebagian besar menggunakan jenis *Universal Product Code* atau UPC. Kode ini terdiri dari 13 angka dengan tiga angka pertama merupakan kode negara barang tersebut dimanufaktur. Untuk Indonesia, kode negaranya adalah 899. Sementara itu, negara lain yang telah memanufaktur berbagai barang seperti Jepang dan Amerika serikat, misalnya, memiliki kode negara lebih dari satu. Contohnya, Amerika Serikat memiliki kode negara 000-139 dan Jepang memiliki kode negara 450-459 dan 490-499. Dengan informasi ini, apabila kita ingin mencari produk dari negara tertentu dengan banyak kode, tentu tidak cukup jika menyebutkan secara

langsung kodenya, karena ada banyak kemungkinan selain kode yang disebutkan.

Bagian selanjutnya merupakan kode unik dari perusahaan manufaktur. Jika kita hanya ingin mencari barang dari negara tertentu saja, bagian ini bisa diabaikan. Tetapi, adakala kita ingin mencari produk tertentu dari perusahaan tertentu, maka baiknya bagian ini ikut dimasukkan dalam pola yang akan dicari. Tidak seperti kode negara, kode perusahaan umumnya hanya satu saja, sehingga dalam melakukan pencarian, kode perusahaan yang ingin dicari cukup ditulis secara utuh. Contoh dari kode perusahaan adalah untuk perusahaan Wings adalah 88666.

Selanjutnya adalah kode unik dari produk. Jika ingin mengambil seluruh produk dari perusahaan tertentu, bagian ini bisa diabaikan. Tetapi, terkadang kita ingin mencari suatu produk tertentu, atau mungkin ingin memeriksa keaslian produk tersebut. Dalam kasus tersebut, menentukan pola untuk kode produk adalah hal yang penting. Kode produk umumnya memiliki suatu pola tertentu, dan pola tersebut bisa berbeda untuk tiap perusahaan. Misalnya, untuk produk perusahaan Wings yang merupakan pembersih berbentuk cairan, kode produk dimulai dengan kode 02 dan diikuti empat digit kode unik produk. Umumnya, digit pertama merupakan kode unik merk dan dua digit terakhir merupakan kode unik varian dari produk tersebut. Misalnya, kode produk dari "So Klin Pewangi Pouch 900ml" adalah 02730 dan kode produk dari "So Klin Pewangi Blue Botol 1 liter" adalah 02761. Dapat kita simpulkan bahwa kode merk untuk "So Klin" adalah 7, dan kode barcode secara keseluruhan untuk kedua produk tersebut adalah 899 88666 02730 dan 899 88666 02761.

Untuk keperluan analisis di makalah ini, akan diambil dua buah kasus kasus. Pertama, akan diperiksa apakah produk berasal dari negara Jerman dengan kode negara 400-440. Kedua, akan diperiksa apakah produk merupakan hasil manufaktur PT Perfetti Van Melle, salah satu perusahaan manufaktur permen di Indonesia dengan kode perusahaan 11150.

B. Aplikasi dengan algoritma Brute Force

Kedua kasus memiliki sedikit perbedaan dalam pencocokan string. Untuk kasus pertama, karena ada empat puluh kemungkinan kode negara, pencocokan menggunakan algoritma Brute Force tidaklah mangkus. Tetapi, karena hanya perlu mencocokan tiga karakter saja, maka penggunaan algoritma Brute Force untuk kasus ini masih dapat dilakukan.

Pada kasus pertama, string teks akan dipotong sehingga kode negara terpisah dengan bilangan lainnya. Hal ini dilakukan untuk mempercepat kinerja pencocokan pola. Setelah itu, akan dilakukan pencocokan dengan string pola yaitu kode negara Jerman. Karena bilangan pertama dari kode negara sudah pasti angka 4, pencocokan dapat dipercepat menjadi dua digit saja.

Pada kasus pertama, misalnya terdapat kasus kode barcode dengan angka 408 25432 11902. Pertama, program akan mencocokkan dengan tujuh pola, karena terdapat empat puluh kemungkinan kode negara untuk Jerman dan kebetulan barcode ini memiliki kode 408. Dalam satu pola, akan dilakukan tiga kali perbandingan, sehingga total telah terjadi 21 perbandingan. Selanjutnya akan dilakukan pencocokan dengan kode 408. Pada kasus ini juga akan terjadi tiga kali perbandingan, sehingga terjadi total 24 kali perbandingan. Kasus terbaik adalah langsung terjadi kecocokan, yaitu apabila kode negara suatu produk adalah 400, atau kode negara suatu produk tidak dimulai dengan angka 4 sama sekali. Sementara itu, kasus terburuk adalah kode negara dimulai dengan angka 4 tetapi bukan merupakan kode negara Jerman. Hal ini menyebabkan diharuskannya pencocokan dengan semua kode, yaitu dimulai dari 400 hingga 440.

Pada kasus kedua, akan digunakan teks yaitu kode 899 11150 65432 untuk analisis. Teks tersebut akan dibandingkan dengan pola yaitu kode perusahaan 11150. Semua barcode dianggap berasal dari Indonesia, sehingga tidak akan ada kode perusahaan yang sama tetapi berbeda negara. Untuk kasus ini, 8 kali perbandingan. Untuk fungsi kedua ini, perlu dilakukan pemeriksaan dimana indeks ditemukannya pola, karena ada kemungkinan pola tersebut ditemukan di bagian kode unik produk. Padahal yang kita cari adalah kecocokan kode barcode dengan kode perusahaan tertentu.

Dari analisis kedua kasus yang telah dilakukan sebelumnya, diperoleh hasil yaitu sebanyak 24 kali perbandingan untuk kasus pertama dan 104 kali perbandingan untuk kasus kedua. Dari hasil tersebut, dapat disimpulkan bahwa algoritma Brute Force tidak mangkus untuk digunakan dalam pemindaian barcode.

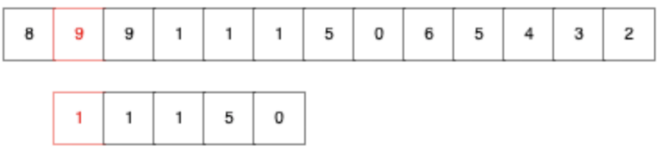
C. Aplikasi dengan algoritma KMP

Untuk kasus pertama, pencocokan pola kode negara Jerman dengan menggunakan algoritma KMP sama seperti algoritma Brute Force, karena hanya dilakukan pengecekan sebanyak tiga karakter saja dan tidak ada pergeseran. Hal ini disebabkan karena string yang diperiksa terletak di awal teks.

Untuk kasus kedua, teks yang digunakan adalah sama seperti sebelumnya, yaitu 899 11150 65432. Pertama, akan dicari *border function* dari pola 11150 dengan hasil sebagai berikut.

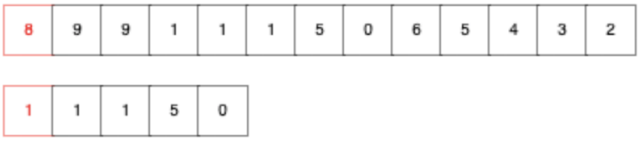
j	0	1	2	3	4
P[j]	1	1	1	5	0
f(j)	0	1	2	0	0

Selanjutnya, akan dilakukan pencocokan string. Berikut ini merupakan ilustrasi pencocokan yang terjadi



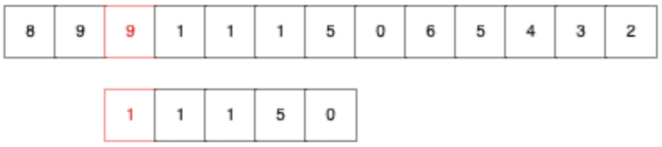
Gambar 3.2.1 Pemeriksaan dengan algoritma KMP iterasi 1

Sumber: Dokumentasi penulis



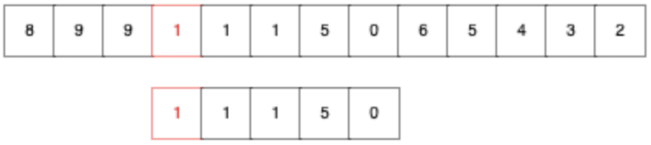
Gambar 3.2.2 Pemeriksaan dengan algoritma KMP iterasi 2

Sumber: Dokumentasi penulis



Gambar 3.2.3 Pemeriksaan dengan algoritma KMP iterasi 3

Sumber: Dokumentasi penulis



Gambar 3.2.4 Pemeriksaan dengan algoritma KMP iterasi 4

Sumber: Dokumentasi penulis

Dapat dilihat bahwa terjadi kecocokan disini, sehingga pola tidak digeser dan pencocokan dilanjutkan ke karakter selanjutnya. Total perbandingan yang terjadi adalah 8 perbandingan. Seperti pada Brute Force, perlu juga diperiksa bahwa kecocokan terjadi pada indeks yang sesuai.

Dapat dilihat bahwa jumlah perbandingan yang terjadi sama dengan algoritma Brute Force. Hal ini disebabkan karena algoritma KMP memang mirip dengan Brute Force, tetapi menggunakan pergeseran yang lebih "pintar". Dalam kasus ini, tidak terjadi pergeseran pintar tersebut.

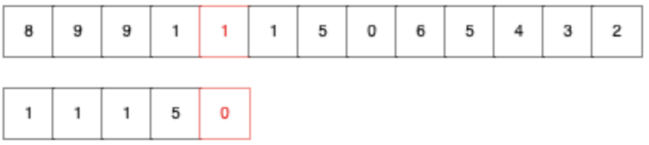
D. Aplikasi dengan algoritma Boyer-Moore

Untuk kasus pertama, pencocokan pola dengan algoritma Boyer-Moore sama seperti algoritma sebelumnya. Alasannya sama, yaitu jumlah karakter yang sedikit serta tidak diperlukan pergeseran.

Untuk kasus kedua, teks yang digunakan adalah sama seperti sebelumnya, yaitu 899 11150 65432. Pertama, akan dicari hasil dari fungsi *last occurrence* dan diperoleh hasil sebagai berikut.

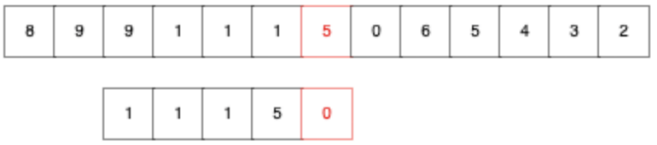
P[j]	0	1	5
l	4	2	3

Selanjutnya akan dilakukan pencarian pola. Berikut ini merupakan ilustrasi pencarian pola dengan menggunakan algoritma Boyer-Moore.



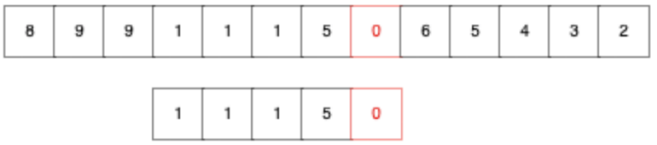
Gambar 3.3.1 Pemeriksaan dengan algoritma Boyer-Moore iterasi 1

Sumber: Dokumentasi penulis



Gambar 3.3.2 Pemeriksaan dengan algoritma Boyer-Moore iterasi 2

Sumber: Dokumentasi penulis



Gambar 3.3.3 Pemeriksaan dengan algoritma Boyer-Moore iterasi 3

Sumber: Dokumentasi penulis

Dapat dilihat bahwa terjadi kecocokan disini, sehingga pencocokan dilanjutkan ke karakter pola selanjutnya. Total perbandingan yang terjadi adalah 7 perbandingan. Pada algoritma ini juga diperlukan pengecekan indeks ditemukannya kecocokan.

Walaupun terlihat tidak jauh, perbedaan disini cukup signifikan jika ada banyak sekali kode yang akan diperiksa. Perlu diingat bahwa panjang string yang dicari polanya juga tidak terlalu besar.

E. Aplikasi regular expression

Untuk kasus pertama, terdapat tiga belas digit kode yang akan dicocokkan. Bagian yang penting hanyalah tiga digit pertama, yaitu kode negara. Ingat kembali bahwa negara Jerman memiliki kode negara 400-440. Perhatikan bahwa digit pertama dari kode negara selalu merupakan angka 4, sedangkan digit terakhir dapat diisi angka apapun yaitu 0-9. Sementara itu, digit kedua hanya dapat diisi angka 0-4 saja. Sisa dari kode barcode dapat diisi bilangan apapun.

Dari penjabaran sebelumnya, kita dapat menyusun sebuah *regular expression* menggunakan sintaks yang ada. Untuk kode negara, *regular expression* yang digunakan adalah `4[0-4]\d`. Ekspresi `[0-4]` artinya posisi tersebut hanya bisa diisi oleh bilangan 0-4, sedangkan `\d` artinya bilangan

tersebut dapat diisi dengan bilangan apapun. Untuk digit sianya, dapat digunakan `(\d){10}` yang artinya dapat diisi oleh bilangan apapun tetapi harus sebanyak sepuluh kali. Tanda kurung disini penting karena untuk memisahkan dengan ekspresi `\d` sebelumnya. Ekspresi lain yang memenuhi adalah `[0-9]{10}`, yang memiliki arti yang sama. Dari penjelasan tersebut kita menemukan bahwa regular expression yang sesuai untuk kasus pertama adalah `4[0-4]\d(\d){10}`.

Untuk kasus kedua, penyusunan regular expression lebih sederhana karena kode negara dan kode perusahaan dicocokkan secara langsung. Pola yang perlu dilihat hanyalah pada lima digit terakhir, yaitu diisi angka sembarang. Hal ini dapat direpresentasikan dengan ekspresi `\d{5}`. Sehingga, dapat diperoleh regular expression untuk kasus kedua adalah `89911150(\d){5}`.

Dari hasil didapat melalui keempat teknik pencocokan pola, dapat disimpulkan bahwa cara yang paling sederhana dan cepat untuk memindai barcode suatu produk spesifik adalah dengan menggunakan *regular expression.*

IV. Kesimpulan

Barcode merupakan hal yang sering kita temui di kehidupan sehari-hari tetapi sering juga kita lewatkan. Ternyata, kode yang tersimpan di barcode bukanlah angka-angka sembarang, melainkan angka identitas suatu produk yang memiliki pola tertentu. Pola tersebut dapat dicari dengan menggunakan teknik pencocokan string sederhana yang sering kita gunakan di dunia Informatika.

Setelah dilakukan analisis dengan berbagai algoritma pencocokan pola, disimpulkan bahwa pencarian pola dalam sebuah kode barcode paling efektif menggunakan *regular expression*. Hal ini disebabkan *regular expression* menyediakan sintaks yang dapat mendeskripsikan suatu pola secara spesifik, sehingga tidak diperlukan pencocokan per karakter. Walaupun begitu, pencocokan per karakter mungkin saja diperlukan untuk beberapa kasus.

Penutup

Penulis mengucapkan terima kasih dan puji syukur kepada Tuhan Yang Maha Esa yang telah memberikan kemudahan sehingga makalah ini dapat selesai dengan tepat waktu. Penulis juga mengucapkan banyak terima kasih kepada:

1. Semua dosen pengajar mata kuliah IF2211 Strategi Algoritma, terutama Bapak Dr. Ir. Rinaldi Munir, M.T., serta Ibu Dr. Nur Ulfa Maulidevi S.T., M.Sc. sebagai dosen pengajar Kelas 02.
2. Keluarga dan teman penulis yang terus mendukung proses pembuatan makalah ini

Penulis merasa bahwa makalah ini masih jauh dari kata sempurna. Oleh karena itu, penulis meminta maaf atas kesalahan yang ditemukan dan segala kritik, saran, serta masukan akan diterima dengan lapang dada. Akhir kata, penulis berharap bahwa makalah ini dapat membawa manfaat baik untuk penulis maupun pembaca.

Referensi

[1] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf, diakses pada 8 Mei 2021

[2] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf, diakses pada 8 Mei 2021

[3] https://corp.trackabout.com/blog/barcodes-brief-history#:~:text=The%20first%20barcode%2C%20with%20a,management%20and%20customer%20check%2Dout., diakses pada 8 Mei 2021

[4] https://www.kiosbarcode.com/blog/anatomi-barcode/, diakses pada 8 Mei 2021

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.

Jakarta, 10 Mei 2021

Nabelanita Utami 13519104