Algoritma *String Matching* dan *Regular Expression* pada Pencarian Judul atau Kode Buku di Perpustakaan

Maximillian Lukman / 13519153
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519153@std.stei.itb.ac.id

***Abstract*—Pada era yang serba modern zaman sekarang, perpustakaan sudah bukanlah menjadi tempat yang relevan di kalangan masyarakat, *gadget* atau alat elektronik lebih dipilih sebagai sumber pencarian dan juga pembelajaran. Meskipun begitu, kebutuhan akan ilmu pengetahuan dan informasi menjadi mutlak untuk dipenuhi agar dapat menghadapi era teknologi informasi saat ini sehingga perpustakaan tetap harus dijaga kualitas dan eksistensinya. Oleh karena itu dibutuhkan banyak fasilitas di sebuah perpustakaan, salah satu fasilitas tersebut adalah mesin pencari buku berdasarkan judul ataupun kode buku tersebut. Mesin ini melakukan pencarian berdasarkan data buku yang ada di perpustakaan tersebut dengan begitu pada dasarnya pencarian tersebut dapat dilakukan menggunakan algoritma pencocokan *string* seperti algoritma Knuth-MorrisPratt (KMP), Boyer-Moore dan *Regular Expression*.**

Keywords—mesin pencari perpustakaan; pencocokan string; KMP; Boyer-Moore; Regular Expression.

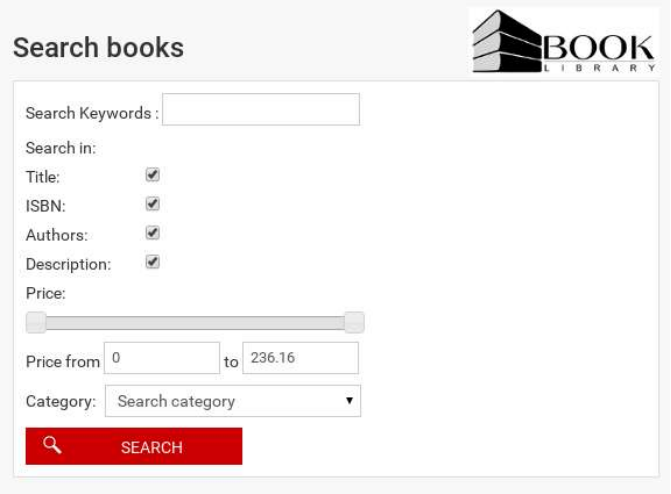
I. PENDAHULUAN

Sekarang ini, ilmu pengetahuan dan teknologi semakin berkembang. Hal ini tentunya membuat kebutuhan manusia tidak hanya antara sandang, pangan, papan lagi tetapi kita juga dituntut untuk selalu mengikuti perkembangan ilmu pengetahuan dan informasi yang ada. Salah satu sarana yang masih digunakan hingga saat ini oleh masyarakat untuk mencari sumber ilmu dan informasi adalah perpustakaan. Meskipun pada zaman sekarang orang-orang umumnya lebih memilih sarana lain seperti mencari sumber langsung dari *gadget* atau alat elektronik yang mereka miliki, perpustakaan masih menjadi solusi dan memiliki nilai lebih apabila yang dicari adalah literasi atau buku-buku yang sulit atau bahkan tidak ada di internet.

Oleh karena itu walaupun sudah terdapat banyak fasilitas atau teknologi pengganti perpustakaan secara umum untuk mencari ilmu dan informasi, perpustakaan tetap harus dijaga kualitas dan eksistensinya dan juga dikembangkan secara terus menerus sehingga tidak tergerus perkembangan zaman dan akhirnya hilang bersama literasi dan buku-buku yang penting yang tidak bisa ditemukan di internet. Meskipun begitu, perpustakaan tetap menjadi pilihan pertama kalangan mahasiswa dalam mencari sumber pembelajaran maupun sumber literasi tertulis untuk makalah dan skripsi mereka.

Perkembangan teknologi pun telah menuntut sistem perpustakaan untuk berubah. Banyak fasilitas-fasilitas baru yang dibutuhkan untuk mengganti sistem konvensional atau yang disebut kuno di suatu perpustakaan. Adanya mesin pencari buku otomatis sebagai salah satu fasilitas merupakan suatu temuan yang sangat bagus untuk mempermudah pengunjung perpustakaan mencari buku berdasarkan judul, penulis, tahun keluaran, maupun kode buku tersebut di data penyimpanan perpustakaan. Mesin ini melakukan pencarian melalui pencocokan pertanyaan atau masukan pengunjung dengan data-data buku yang ada di basis data perpustakaan.

Pada dasarnya, metode pencarian buku ini dilakukan dengan memanfaatkan sebuah algoritma yang disebut algoritma *string matching* atau pencocokan *string*. Mesin tersebut menerima masukan pengunjung dan kemudian mencocokkan kata demi kata sesuai dengan algoritma yang dipakai. Algoritma yang paling terkenal dan akan dibahas di sini adalah algoritma Knuth-MorrisPratt (KMP), algoritma Boyer-Moore, dan juga penggunaan *Regular Expression*.



Gambar 1.1 Contoh Mesin Pencarian Buku di Perpustakaan
Sumber : https://ordasoft.com/
(diakses pada 7 Mei 2021, 19.15)

II. TEORI DASAR

A. *Algoritma Pencocokan String*

Algoritma ini merupakan algoritma yang dilakukan untuk menemukan seluruh kemunculan rangkaian string atau *pattern* dalam sebuah string yang lain dimana string atau pattern yang dicari memiliki panjang yang lebih sedikit atau pendek dibandingkan dengan string/teks yang dicari isinya untuk ditemukan pola string yang menjadi bahan pencarian.

Berikut adalah beberapa istilah yang harus diketahui dalam string matching:

a. *Teks* (T), menyatakan sekumpulan kata atau kalimat yang akan diuji, dalam hal ini teks menyatakan data-data dari buku yang telah disimpan di perpustakaan. Panjang teks dilambangkan dengan n.

b. *Pattern* (P), menyatakan string yang akan dicari di dalam teks. Dalam hal ini panjang pattern dilambangkan dengan m dan diasumsikan m <<< n atau m jauh lebih kecil daripada n.

c. *Prefix*. Jika S adalah sebuah substring, S[1...k-1] adalah prefix dari S.

d. *Suffix*. Jika S adalah sebuah substring, S[k-1...m] adalah suffix dari S.

e. k merupakan sebuah index diantara 1 sampai m.

Banyak sekali algoritma *string matching* yang ada seperti algoritma *straightforward matching*/*brute-force*, *finite automata*, Rabin-Karp, Knuth-MorrisPratt(KMP), Boyer-Moore, dll. Tetapi pada karya ilmiah ini, algoritma yang akan dipakai adalah algoritma-algoritma yang paling terkenal yaitu algoritma *brute-force*, Knuth-MorrisPratt(KMP), Boyer-Moore, dan *Regular Expression*.

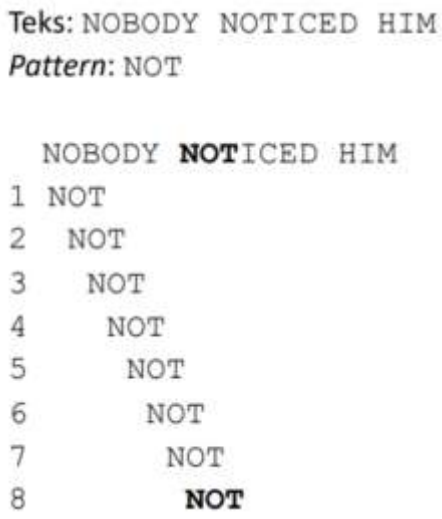
B. *Algoritma Brute-Force*

Algoritma ini juga dinamakan *naïve string matching* karena teknik yang digunakan terlalu *simple-minded* dan memakan waktu yang sangat lama tetapi paling mudah dimengerti. Algoritma ini digunakan tanpa memerhatikan kinerja atau performa dari suatu sistem. Algoritma ini bekerja dengan cara menelusuri teks dan mencocokkannya dengan pattern hingga ditemukan pattern yang cocok pada teks. Algoritma akan melakukan pencocokan satu per satu pada setiap posisi didalam teks.

Misalnya terdapat sebuah teks (T) dan sebuah pattern (P) yang ingin dicari kemunculannya. Langkah-langkah algoritma *brute-force* adalah sebagai berikut:

1. Pencocokan karakter pertama pada pattern P dan teks T.
2. Apabila karakter sama, maka karakter berikutnya pada pola akan dicek dengan karakter berikutnya pada teks.
3. Pengecekan dilakukan untuk setiap karakter P dengan teks T sampai seluruh karakter P yang dicocokkan dengan T sudah sama atau terdapat ketidakcocokan pertama antara karakter di P dan teks T.
4. Apabila terdapat ketidakcocokan, maka karakter pertama dari P akan digeser satu karakter dan dicek kembali dari depan dengan karakter berikutnya pada T.

Berikut ini adalah contoh penggunaan algoritma *brute-force* beserta dengan *pseudocode*-nya:



Gambar 2.1 Contoh Penggunaan Algoritma Brute-Force
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
(diakses pada 7 Mei 2021, 19.45)

```
function bruteForce (input P: array[1...m] of
char, T: array[1..n] of char)

P: array[1..m] of char //string pattern yang
akan dicari
T: array[1..n] of char //string teks
m: panjang pattern
n: panjang teks

for i ← 0 to n-m do
      j ← 0
      while (j < m) and (T[i+j] = P[j]) do
            j ← j+1
      if (j = m) return i
return -1
```

Adapun kompleksitas algoritma pencocokan string secara brute-force adalah sebagai berikut tergantung dari kasus-kasus yang ditemukan dalam pencarian:

Kasus	Kompleksitas Waktu
Best Case	O(n)
Average Case	O(m+n)
Worst Case	O(mn)

Tabel 2.1 Kompleksitas Waktu Algoritma Brute-Force

C. Algoritma Knuth-Morris-Pratt (KMP)

Algoritma Knuth-Morris-Pratt pertama kali ditemukan oleh seorang ilmuwan komputer sekaligus profesor di Stanford University yaitu Donald Erwin Knuth bersama dengan J. H. Morris dan V. R. Pratt. Algoritma KMP ini memiliki cara kerja yang mirip dengan algoritma Brute-Force tetapi dengan pencocokan string yang lebih pintar dan lebih efektif dengan menyimpan informasi yang digunakan untuk melakukan jumlah pergeseran tetapi tetap dilakukan secara terurut dari kiri ke kanan. Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter per karakter seperti pada algoritma Brute Force. Yang membedakan adalah algoritma ini menggunakan prefix dan suffix pada pola untuk melakukan pencocokan string pola pada teks. Panjang prefix dan suffix yang sama pada sebuah pola akan dicatat yang biasa disebut dengan *border function*.

Border function atau fungsi pinggiran didefinisikan sebagai jumlah karakter maksimal pada prefix dari pattern P[1..k] yang juga merupakan suffix dari pattern. Border function ini hanya bergantung pada karakter-karakter yang terdapat pada pattern, sehingga dapat dilakukan perhitungan fungsi tersebut sebelum pencarian *string* dilakukan. Dalam menggunakan algoritma KMP ini, kita perlu membuat sebuah tabel yang disebut tabel fungsi pinggiran yang kita jadikan patokan kemana kita harus menggeser pattern yang kita cari pada teks, penentuan tabel tersebut menggunakan bantuan prefix dan suffix dari pattern. Misalkan terdapat contoh yaitu pattern P = abaaba. Nilai fungsi pinggiran b(j) untuk tiap karakter pada P adalah sebagai berikut:

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

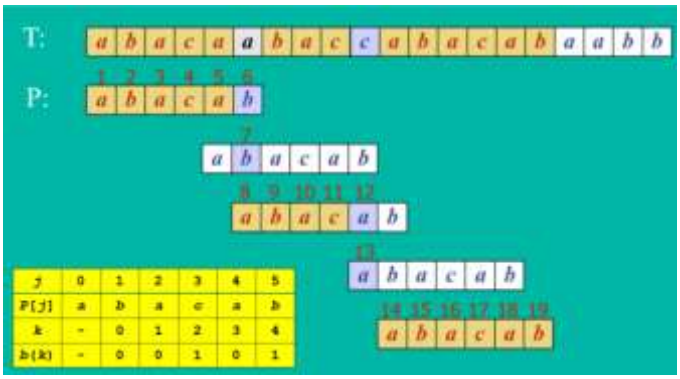
Tabel 2.2 Border Function dari Pattern “abaaba”

Berikut ini adalah cara kerja dari algoritma KMP adalah sebagai berikut:

1. Pencocokan karakter pertama pada pattern dan teks.
2. Apabila karakter sama, pencocokan maju ke karakter berikutnya untuk dicek karakter berikutnya pada pattern dengan karakter berikutnya pada teks.
3. Apabila terdapat ketidakcocokan, ambil fungsi pinggiran dari karakter sebelumnya lalu tambahkan dengan 1 dan itulah indeks karakter berikutnya yang akan diperiksa.
4. Langkah diatas diulangi terus menerus sampai diperoleh pattern yang sesuai pada teks

Berdasarkan langkah-langkah diatas, dapat dilihat bahwa algoritma Knuth-Morris-Pratt atau KMP ini dapat mempercepat waktu dalam pencarian pattern karena tidak harus memeriksa kembali seluruh karakter sepanjang pattern melainkan hanya melakukan pengecekan pada karakter yang masih mungkin saja. Selain itu, proses juga tidak perlu melakukan penggeseran mundur seperti yang dilakukan pada algoritma Brute-Force, tetapi jika ukuran dari banyaknya huruf meningkat, maka algoritma ini dapat menjadi tidak efektif.

Berikut ini adalah contoh penggunaan algoritma Knuth-Morris-Pratt dan juga contoh programnya dalam bahasa Java:



Gambar 2.2 Contoh Penggunaan Algoritma Knuth-Morris-Pratt

Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
(diakses pada 7 Mei 2021, 19.45)

```
public static in kmpMatch (String text,
String pattern)
{
     int m = pattern.length();
     int n = text.length();
     int fail[] = computeFail(pattern);
     int i=0;
     int j=0;
     while (i < n) {
        if (pattern.charAt(j) ==
        text.charAt(i)) {
           if (j == m - 1) {
              return i - m + 1; // pattern
ditemukan
           }
           i++;
           j++;
        }
        else if (j > 0) {
           j = fail[j-1];
        }
        else {
          i++;
        }
     }
     return -1; // pattern tidak ditemukan
} // end of kmpMatch()
```

```
public   static   int[]   computeFail(String
pattern)
{
   int fail[] = new int[pattern.length()];
   fail[0] = 0;
   int m = pattern.length();
   int j = 0;
```

```
    int i = 1;
    while (i < m) {
        if (pattern.charAt(j) ==
        pattern.charAt(i)) {
            fail[i] = j + 1;
            i++;
            j++;
        }
        else if (j > 0) {
            j = fail[j-1];
        }
        else {
            fail[i] = 0;
            i++;
        }
    } return fail;
} // end of computeFail()
```

D. Algoritma Boyer-Moore

Algoritma Boyer-Moore pertama kali ditemukan oleh Bob Boyer dan J. S. Moore pada tahun 1977. Algoritma ini memiliki langkah yang lebih kompleks dibandingkan dengan algoritma sebelumnya karena terdapat tiga kasus dalam setiap langkah yang dilakukan. Meskipun begitu, algoritma Boyer-Moore dianggap oleh kalangan praktisi sebagai algoritma pencocokan string paling efisien pada penggunaan string matching pada umumnya dan juga banyak digunakan untuk mencocokkan pattern tertentu pada teks yang panjang. Pencocokan string pada algoritma ini dilakukan dari pola kanan ke kiri pada pattern.

Pada algoritma Boyer-Moore dikenal istilah *last occurrence* yang merupakan posisi terakhir suatu karakter pada pattern. Karakter yang dilihat *last occurrence*-nya adalah karakter pada teks yang pengujian kesamaannya bernilai gagal. Algoritma ini melakukan dua hal yaitu dia teknik looking glass dan juga character jump yang menyebabkan dia menjadi algoritma yang lebih efisien. Teknik-teknik ini saling membantu membuat algoritma menjadi sangat efisien untuk melakukan pencarian string di dalam teks. Kedua teknik ini adalah sebagai berikut:

a. Teknik *looking-glass*

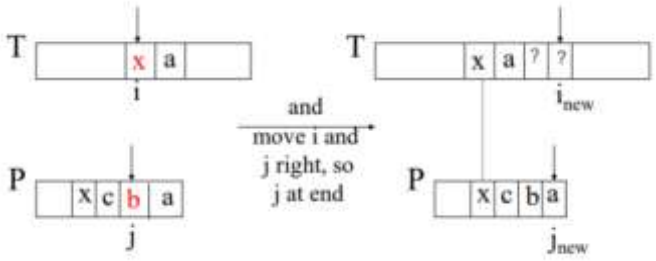
   Teknik ini mencari pattern pada teks dengan menguji karakter pada P dari belakang atau indeks tertinggi.

b. Teknik *character-jump*

   Apabila terjadi ketidakcocokan dalam pengujian karakter, maka dilakukan pergeseran karakter.

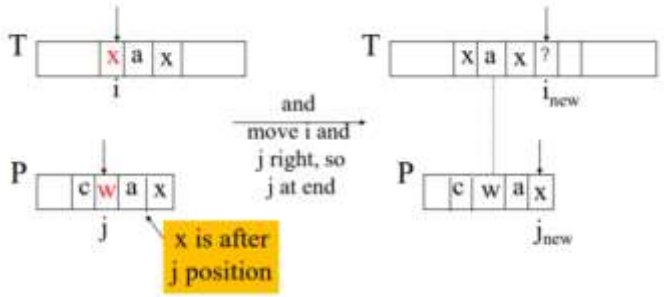
Pada teknik *character-jump* terdapat tiga buah kasus dalam penggeseran pattern terhadap teks. Ketiga kasus tersebut berhubungan dengan keberadaan karakter pada pattern. Kasus-kasus itu adalah sebagai berikut:

- Kasus pertama, terjadi ketika *last occurrence* terdapat pada sisi kiri karakter yang sedang dicek pada pattern. Pada kasus ini, pattern digeser ke kanan sehingga karakter yang bersesuaian berada pada posisi yang sejajar.



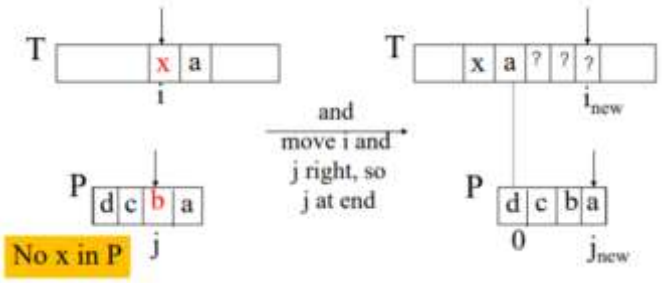
Gambar 2.3 Kasus Pertama Character-Jump
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
(diakses pada 7 Mei 2021, 19.45)

- Kasus kedua, terjadi jika posisi *last occurrence* karakter yang diuji pada teks terdapat pada sebelah kanan karakter yang diuji pada pattern. Pada kasus ini, pattern digeser ke kanan satu karakter.



Gambar 2.4 Kasus Kedua Character-Jump
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
(diakses pada 7 Mei 2021, 19.45)

- Kasus ketiga, terjadi jika karakter yang diuji pada teks tidak terdapat pada pattern. Pada kasus ini, pattern digeser ke kanan sampai karakter awal pattern sejajar dengan satu karakter sebelah kanan karakter teks yang diuji.



Gambar 2.5 Kasus Ketiga Character-Jump
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
(diakses pada 7 Mei 2021, 19.45)

Berikut ini adalah contoh penggunaan algoritma Boyer-Moore beserta potongan program dalam bahasa Java:

Gambar 2.6 Contoh Penggunaan Algoritma Boyer-Moore
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Pencocokan-string-2021.pdf
(diakses pada 7 Mei 2021, 19.45)

```
public static int bmMatch(String text,
String pattern)
{
   int last[] = buildLast(pattern);
   int n = text.length();
   int m = pattern.length();
   int i = m-1;
   if (i > n-1)
      return -1
   int j = m-1;
   do {
      if (pattern.charAt(j) ==
      text.charAt(i)) {
         if (j == 0){
            return i; // match
         }
         else { // looking-glass technique
            i--;
            j--;
         }
      }
      else { // character jump technique
         int lo = last[text.charAt(i)];
         i = i + m - Math.min(j, 1+lo);
         j = m - 1;
      }
   } while (i <= n-1);
   return -1; // pattern tidak ditemukan
} // end of bmMatch()
```

E. Regular Expression

Regular Expression atau biasa disebut sebagai *regex* merupakan sebuah algoritma atau alat untuk melakukan pencarian pola tertentu yang cocok pada sebuah atau lebih karakter pada string panjang/teks. Regex ini memiliki kelebihan yaitu pencocokan yang dilakukan tidak hanya harus pada sebuah pola yang statis atau spesifik, tetapi juga dapat mencocokkan pola-pola lain yang mirip dan juga satu himpunan karakter.

Sintaks dari regular expression berupa kumpulan karakter-karakter khusus yang menentukan pola pencarian. Ekspresi regex biasanya disebut dengan pattern. Sintaks-sintaks tersebut terdiri atas beberapa aturan. Aturan-aturan tersebut adalah sebagai berikut:

.	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*.

Gambar 2.7 Aturan Penggunaan Regular Expression
Sumber :
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf
(diakses pada 7 Mei 2021, 20.00)

III. ANALISIS

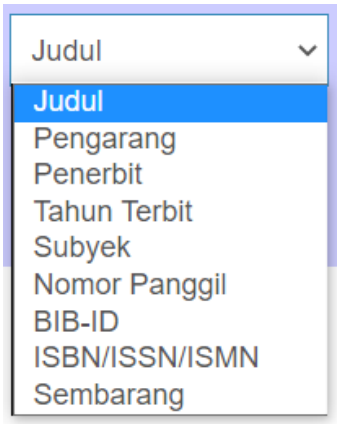
Pada percobaan perbandingan yang dilakukan, mesin pencarian buku yang akan digunakan sebagai contoh adalah mesin pencarian dari website Online Public Access Catalog milik Perpustakaan Nasional RI (https://opac.perpusnas.go.id/)

Gambar 3.1 Tampilan Online Public Access Catalog Perpustakaan Nasional RI

A. Sistem Pencarian Buku

Perpustakaan nasional pastinya menggunakan database/basis data yang cukup besar untuk menyimpan seluruh data dan informasi dari buku yang dimiliki dan tersedia di perpustakaan. Atribut-atribut yang ada pada basis data pastinya disesuaikan dengan kategori query apa saja yang bisa pengguna masukkan seperti apabila pengguna ingin mencari buku berdasarkan judul, pengarang, penerbit, tahun terbit, nomor ISBN, dll.

Pada website Online Public Access Catalog tersebut, terdapat beberapa kategori yang pengguna bisa gunakan sebagai dasar pencarian yaitu sebagai berikut:



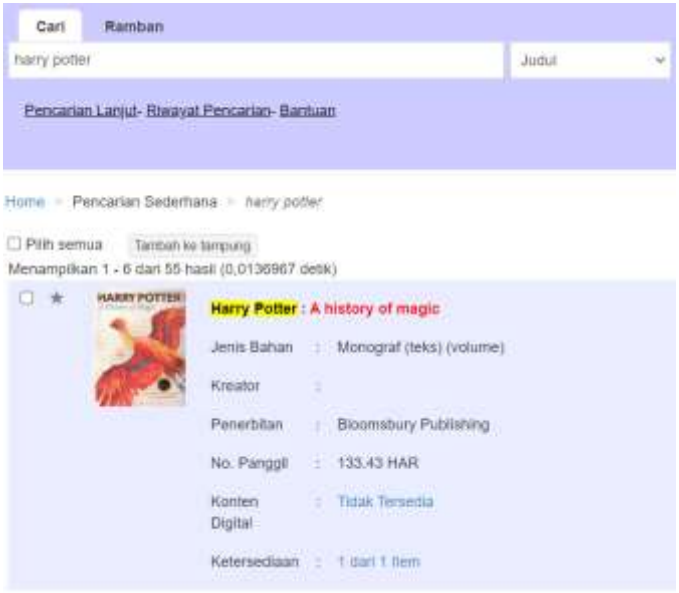
Gambar 3.2 Kategori Pencarian Buku

Dari apa yang saya pelajari mengenai string matching dan regex serta setelah menggunakannya pada salah satu tugas besar yang saya dapat pada mata kuliah Strategi Algoritma, query atau masukan yang diberi pengguna harus diseleksi atau di-*parsing* terlebih dahulu dan setelah itu akan dicari atau *fetching* data yang diminta pengguna kepada database. Apabila tidak terdapat ciri khas khusus untuk membedakan isi dari query pengguna seperti tanda petik maka mesin tidak akan tahu apa yang ingin pengguna cari, tetapi apabila diberikan kategori-kategori seperti diatas, tidak diperlukan lagi ciri khas khusus tersebut sehingga pencarian dilakukan berdasarkan jenis kategori yang dipilih dan apabila memilih kategori sembarang, mesin akan mencari atau mengecek seluruh atribut dari basis data apakah ada string yang cocok dengan query pengguna.

B. Aplikasi Algoritma String Matching dan Regex

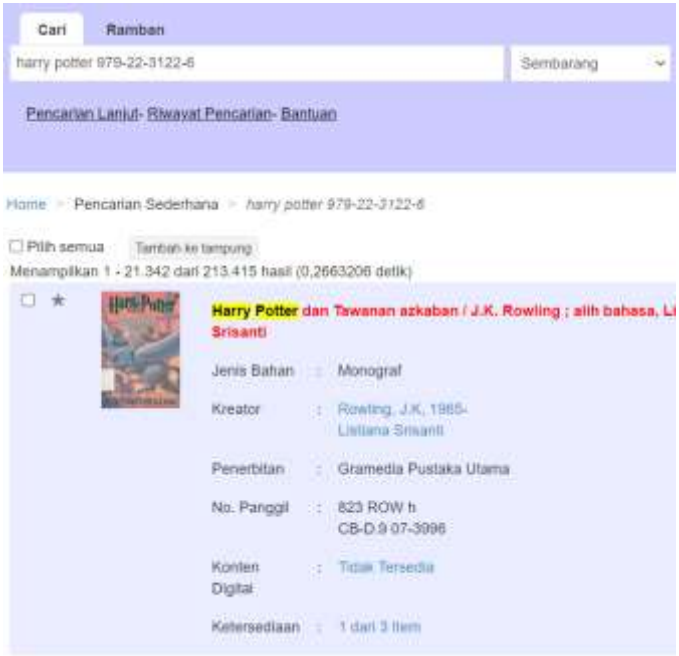
Apabila melihat sistem pencarian pada website Online Public Access Catalog milik Perpustakaan Nasional RI, dapat dibilang bahwa mesin untuk pencarian buku tersebut menggunakan salah satu algoritma string matching dan juga regular expression dalam memproses query input pengguna dan mencocokkannya dengan database yang dimiliki perpustakaan.

Kemungkinan pengambilan data yang dilakukan adalah dengan menyimpan kata-kata dari query pengguna dan dipisahkan menurut kategori. Misalnya input pengguna adalah "harry potter" dan kategori yang dipilih untuk dicari adalah judul, keluarannya adalah sebagai berikut:



Gambar 3.3 Hasil Pencarian String "harry potter"

Dapat dilihat bahwa query pengguna "harry potter" pastinya hanya dicocokkan dengan atribut judul pada database. Tetapi apabila kita memilih kategori sembarang dan memasukkan juga nomor ISBN salah satu buku yaitu "Harry Potter dan Tawanan Azkaban" secara spesifik, maka pencarian akan diurutkan berdasarkan kecocokkan data yang paling banyak yaitu disini judul beserta nomor ISBN-nya.



Gambar 3.4 Hasil Pencarian String "harry potter" dan Nomor ISBN

Adapun jika kita memasukkan "harrypotter" tanpa spasi dan dengan kategori judul, mesin pencari tidak akan menampilkan hasil apa-apa karena tidak ada satu pun judul buku dengan string "harrypotter".

Gambar 3.4 Hasil Pencarian String "harrypotter" Tanpa Spasi

Pada saat melakukan pengecekan pada input yang dimasukkan pengguna, disitulah terjadi string matching. Dapat dilihat dari hasil-hasil diatas bahwa pencarian juga tidak menghiraukan huruf besar atau tidak *case-sensitive*.

Jika melihat dari algoritma-algoritma string matching yang telah dibahas pada Bab 2, tidak mungkin mesin pencarian ini menggunakan algoritma brute-force untuk mencocokkan query pengguna karena karakter harus dicocokkan satu per satu dan akan memakan waktu yang sangat lama, terlebih jika menemukan *worst-case* dimana karakter yang tidak cocok hanya di akhir karena bisa saja pengguna salah dalam memasukkan query dan menimbulkan typo di akhir query.

Oleh karena itu, mesin pencarian ini pasti menggunakan algoritma-algoritma yang lebih efisien seperti algoritma KMP dan Boyer-Moore yang juga sudah dibahas sebelumnya karena pencarian akan lebih cepat dan pengecekkan karakter tidak perlu satu per satu. Pengecekan dilakukan dengan cara membandingkan suatu kata dari query dengan masing-masing tuple dari atribut kategori yang dimaksud pengguna.

Selain itu, penggunaan regular expression pada dasarnya hanya untuk menyeleksi atau *parsing* query pengguna dan menentukan apakah yang dicari adalah judul, tanggal terbit, penerbit, nomor ISBN, dsb. Tetapi karena pada mesin pencari tersebut telah disediakan kategori-kategori, maka regex sebenarnya tidak begitu dibutuhkan, hanya saja apabila kategori yang dipilih adalah "Sembarang" maka pencarian tidak didasarkan pada apa-apa dan dicari pada keseluruhan tabel tanpa mengetahui apa yang ingin dicari pengguna. Oleh karena itu dibutuhkanlah regex untuk setidaknya mesin dapat menentukan apa yang ingin dicari pengguna berdasarkan kata-kata yang terdapat pada query. Misalnya pada gambar 3.4, dipilih kategori sembarang dan memasukkan sederetan angka yang sebenarnya adalah nomor ISBN.

Misalkan terdapat nomor ISBN sebagai berikut:

- 979-22-3122-6
- 978-979-22-3122-6

Terdapat banyak format untuk nomor ISBN seperti pada contoh diatas, tetapi sebenarnya semua nomor ISBN memiliki pola yang sama yaitu angka ditambah dengan tanda *dash* (-) di setiap interval tertentu. Oleh karena itu, kita dapat menggunakan regular expression untuk menyeleksi nomor ISBN tersebut dari query pengguna. Contoh dari regular expression yang dapat digunakan adalah sebagai berikut:

/((\d+)-)*(\d+)/g

dimana apabila terdapat sejumlah angka diikuti simbol *dash* yang dapat berulang 0 kali atau lebih dan diakhiri dengan sejumlah angka lagi, string tersebut akan digolongkan sebagai nomor ISBN oleh program dan akan mengambil atau *fetch* data dari atribut nomor ISBN pada database sehingga pencarian lebih mudah dan cepat untuk dilakukan.

IV. Kesimpulan

Dari karya ilmiah ini, saya menyadari bahwa algoritma string matching ini memiliki banyak sekali kegunaan, tidak terbatas hanya pada search engine ataupun mesin pencari seperti ini. Khususnya pada kasus seperti ini, algoritma string matching seperti algoritma KMP dan Boyer-Moore dapat menjadi solusi yang memudahkan kita dalam mencari sebuah data yang berada pada suatu tempat penyimpanan atau database skala besar karena apa yang dicari (string) dapat ditemukan dengan sangat cepat tanpa harus mengeceknya satu per satu seperti apabila menggunakan algoritma Brute-Force. Terlebih lagi apabila kita juga menerapkan fungsi Regular Expression pada saat *parsing* query input pengguna sehingga apa yang ingin dicari atau dituju dapat lebih cepat dikenali.

V. Penutup dan Ucapan Terima Kasih

Penulis mengucapkan syukur dan terimakasih kepada Tuhan yang Maha Esa karena berkat rahmat-Nya-lah penulis mampu mendapatkan tema dan menyelesaikan karya ilmiah ini dengan baik dan tepat waktu. Penulis juga ingin mengucapkan terimakasih yang sebesar-besarnya kepada semua pihak yang telah membantu dalam pembuatan karya ilmiah "Algoritma *String Matching* dan *Regular Expression* pada Pencarian Judul atau Kode Buku di Perpustakaan" ini, khususnya kepada Bapak Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc., Ph.D. selaku pembimbing dan dosen mata kuliah IF2211 Strategi Algoritma – Semester II Tahun 2020/2021 yang telah mengajar dan membuat penulis dapat menyelesaikan tugas ini dengan baik.

Link Video Youtube

https://youtu.be/jhr4IWeMsb8

Referensi

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

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

[3] https://ordasoft.com/book-search-in-book-library-ebook-management-software-for-library-management

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, 8 Mei 2021

Maximillian Lukman - 13519153