Perbandingan Algoritma Boyer-Moore dan Turbo Boyer-Moore dalam Query MySQL

Margaretha Siahaan - 13509086
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509086@std.stei.itb.ac.id

Makalah ini membahas perbandingan kecepatan penelusuran basis data berdasarkan *query* SQL dengan menggunakan algoritma Boyer-Moore dan algoritma Turbo Boyer-Moore. Dalam pengaksesan basis data dibutuhkan kecepatan yang tinggi agar proses yang menggunakan informasi tersebut dapat berjalan efektif sehingga dibutuhkan algoritma pencarian *string* yang cepat dalam penelusuran *query*. MySQL merupakan salah satu Sistem Manajemen Basis Data Relasional yang sering digunakan untuk aplikasi-aplikasi sederhana. Untuk pencarian pada basis data dengan *query* SQL diperlukan algoritma pencarian *string* yang sesuai. Beberapa algoritma pencarian *string* adalah algoritma Boyer-More dan algoritma Turbo Boyer-Moore. Setelah dilakukan perbandingan antara kedua algoritma tersebut didapatkan bahwa untuk perintah query seperti LIKE, algoritma Turbo Boyer-Moore dapat melakukan lebih sedikit perbandingan dalam penelusurannya.

***Index Terms*— pencarian string, Boyer-Moore, Turbo Boyer-Moore, SQL, mySQL.**

I. PENDAHULUAN

Secara umum, kebanyakan perangkat lunak menggunakan sistem basis data mandiri yang digunakan oleh pengguna untuk menyimpan dan melihat kembali informasi yang diinginkan. Kebanyakan perangkat lunak seperti ini digunakan untuk kegiatan bisnis, pemerintahan, atau kegiatan administrasi lainnya. Untuk tujuan penggunaan yang selalu mengalami perubahan, misalnya bisnis, basis data yang digunakan tentu harus dapat diakses dengan cepat. Selain itu, proses pengaksesan tersebut tidak boleh memberatkan perangkat keras yang digunakan, sehingga sistem tidak terganggun dan proses bisnis tetap berjalan.

Untuk membangun basis data diperlukan sebuah *Database Manageneme*nt *System* (DBMS) yang dapat membantu aplikasi dalam melakukan pengaksesan basis data. Untuk penggunaan basis data yang sederhana MySQL merupakan salah satu pilihan yang dapat digunakan oleh para developer perangkat lunak.

Pada salah satu dokumentasinya, MySQL menuliskan bahwa pencarian string dengan *keyword* **LIKE** pada *query* akan dilakukan dengan algoritma Turbo Boyer-Moore dan bukan dengan menggunakan algoritma pencarian string yang digunakan untuk pencarian *full text* biasa.

Karena algoritma pencarian *string full text* yang digunakan oleh MySQL tidak dapat ditemukan dari sumber-sumber yang ada, maka dilakukan perbandingan antara algoritma Turbo Boyer-Moore dengan algoritma yang paling dekat dengannya, yaitu algoritma Boyer-Moore.

II. LANDASAN TEORI

A. MySQL dan Query SQL

MySQL adalah salah satu aplikasi *Database Management System* (DBMS) yang menggunakan fungsi relasional dalam pengaksesan basis data. MySQL berjalan sebagai sebuah *server* sehingga akses basis data dapat dilakukan oleh beberapa pengguna sekaligus. MySQL dapat berjalan dengan baik dengan semua sistem operasi dan dapat diimplementasikan dengan berbagai API (*Application Programming Interface*) yang berjalan dalam bahasa C, C++. C#, Java, dan lainnya.

Query yang digunakan pada MySQL sama dengan *query* relasional pada umumnya. Terdapat tiga perintah utama dalam operasi pengaksesan basis data yaitu:

- **SELECT**
  Digunakan untuk mengambil data dari basis data.
- **FROM**
  Digunakan untuk memilih tabel yang menyimpan data yang ingin diakses.
- **WHERE**
  Digunakan untuk memberikan kondisi yang membatasi pemilihan data yang akan diakses.

Contoh penggunaan query untuk pengaksesan database adalah sebagai berikut.

```
SELECT *
FROM nasabah
WHERE jumlah_rek > 2000000 AND
kota = 'Bandung'
```

Pada *query* di atas akan dilakukan pengaksesan untuk semua data nasabah yang nilai nominal rekeningnya lebih dari 2000000 dan berada di Bandung.

Pada klausa **WHERE** digunakan berbagai perintah relasional, seperti **JOIN**, **LIKE**, dan perintah operasi

aritmatika relasional lainnya. Dengan perintah **LIKE**, akan dicari semua *string* pada basis data yang memiliki bagian seperti *string* yang mengikuti kata kunci **LIKE** tersebut (*pattern*). *String* yang akan dicari dapat menggunakan dua pilihan operator pembatas:

- '%' digunakan untuk mencocokkan string dengan panjang yang berbeda (termasuk 0)
- '_' digunakan untuk mencocokkan satu karakter pada string.

Contoh *query* penggunaan perintah **LIKE** adalah sebagai berikut.

```
SELECT *
FROM nasabah
WHERE nama_nasabah like 'Mic%'
```

Pada *query* di atas akan dilakukan pengaksesan semua data nasabah yang namanya diawali dengan 'Mic'.

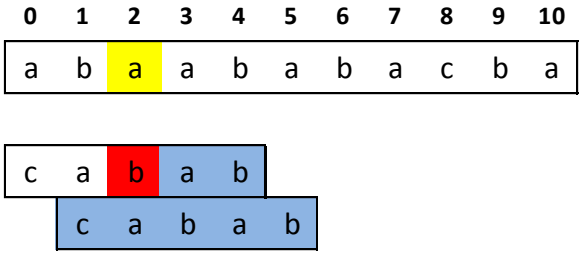
B. Algoritma Pencarian String Boyer-Moore

Algoritma Boyer-Moore merupakan salah satu algoritma yang dirasa paling efisien untuk pencocokan string dalam bahasa natural sehingga sangat sering dipakai untuk berbagai *text editor*. Dalam pseudocode sederhana, algoritma ini ditulis seperti berikut.

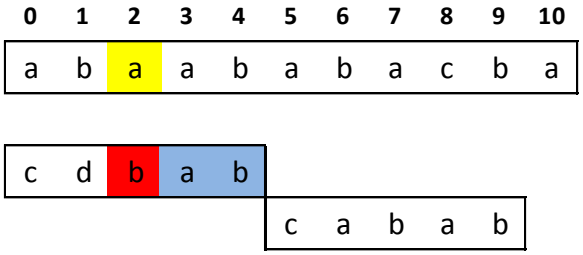
```
function BMSearch(pattern, text : string) → integer
m,n : integer
m ← pattern.length();
n ← text.length();
s := 0 (* the current shift *)
while s <= n - m do {
  j := m - 1
  while P[j] = T[s+j] and j >= 0 do j := j - 1
  if j < 0 return s
 s := s + max(1, j - last[T[s+j]] + 1)
```

Pencarian *string* dilakukan dengan mencocokkan karakter paling kanan pada *pattern* dengan teks pada indeks yang bersesuaian. Jika terjadi ketidakcocokan, maka akan *string* pada teks yang tidak cocok tadi (disebut juga *bad-character*) akan dicari keberadaannya pada *pattern* yang sedang dicari. Jika ada lebih dari satu karakter yang sama, maka diambil karakter yang letaknya paling kanan. *Pattern* akan digeser sampai karakter paling kanan yang sama tersebut berada di bawah *bad-character* tadi. Setelah itu dilakukan pencocokan lagi dari karakter terakhir pada *pattern*.

Ilustrasi penelusuran dengan algoritma Boyer-Moore yang menggunakan metode *bad-character* adalah sebagai berikut.



Gambar 1 Pencarian pattern yang bad-character pada teks berada di pattern

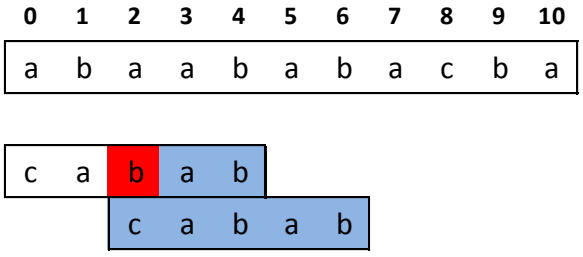


Gambar 2 Pencarian pattern yang bad-character tidak pada teks berada di pattern

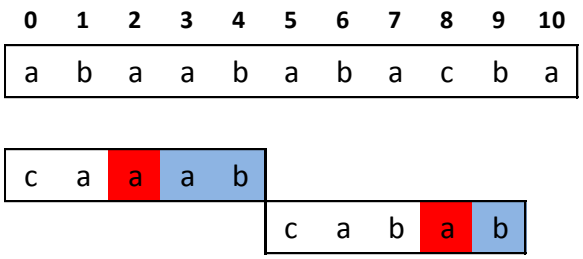
Pada ilustrasi di atas, karakter berwarna merah adalah karakter pada *pattern* yang mengalami ketidakcocokan, karakter berwarna kuning adalah *bad-character* yang ada pada teks, dan karakter biru adalah karakter *pattern* yang sudah cocok dengan karakter pada teks.

Metode pencarian lain adalah dengan mencari *substring* yang mirip pada *pattern* (disebut juga *good-suffix*). Dengan metode ini, jika terjadi kesalahan dan *suffix* yang sudah cocok sebelumnya memiliki *substring* yang sama pada *prefix*-nya, maka *pattern* akan digeser sampai *substring* yang sama tadi tersusun pada indeks yang sesuai. Cara ini agak mirip dengan algoritma Knuth-Morris-Pratt, namun lebih mudah dan cepat untuk bahasa natural karena pencariannya dimulai dari bagian akhir *pattern*.

Ilustrasi penelusuran dengan algoritma Boyer-Moore yang menggunakan metode *good-suffix* adalah sebagai berikut.



Gambar 3 Pencarian pattern yang memiliki good-suffix



Gambar 4 Pencarian pattern yang tidak memiliki good-suffix

Pada ilustrasi di atas, karakter berwarna merah adalah karakter pada *pattern* yang mengalami ketidakcocokan dengan karakter pada teks dan karakter berwarna biru adalah karakter *pattern* yang cocok dengan teks.

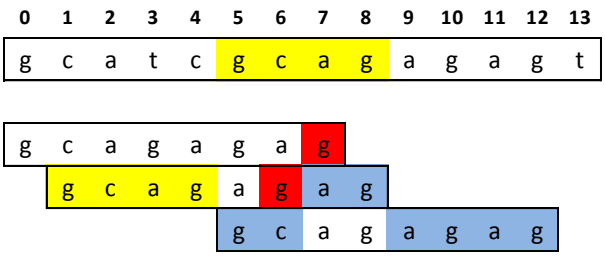
Kompleksitas waktu pencarian algoritma Boyer-Moore adalah **O**(mn) dan kompleksitas waktu pra-proses pencarian adalah **O**(m+σ). Untuk kasus terburuk akan terjadi 3*n* kali perbandingan karakter pattern dan teks yang ditelusuri.

C. Algoritma Pencarian String Turbo Boyer-Moore

Algoritma ini merupakan pengembangan dari algoritma Boyer-Moore. Pada algoritma ini dilakukan pencatatan segmen dari teks yang cocok dengan sebuah suffix pada pattern yang terakhir dilakukan pencocokan. Keuntungan dari penggunaan algoritma ini adalah dimungkinkan terjadinya 'lompatan' melewati segmen tersebut dan dapat melakukan *turbo-shift*.

Turbo-shift dapat terjadi jika pada terdapat *substring* dari *pattern* yang sama dengan *substring* pada teks yang sudah diperiksa sebelumnya.

Ilustrasi penelusuran dengan algoritma Turbo Boyer-Moore adalah sebagai berikut.



Gambar 5 Pencarian pattern dengan algoritma Turbo Boyer-Moore

Pada ilustrasi di atas, karakter berwarna merah adalah karakter pada *pattern* yang mengalami ketidakcocokan dengan karakter pada teks, karakter berwarna biru adalah karakter *pattern* yang cocok dengan teks, dan *substring* kuning adalah karakter-karakter (*substring*) pada teks yang sudah diperiksa dan sama dengan sebuah *substring* dari *pattern*.

Algoritma Turbo Boyer-Moore merupakan variasi dari algoritma Boyer-Moore yang tidak memerlukan pra-proses tambahan yang ada pada algoritma Boyer-Moore. Kompleksitas waktu pada fase pencarian adalah **O**(n) dan untuk kasus terburuk akan terjadi 2*n* kali perbandingan karakter.

Algoritma Turbo Boyer-Moore melakukan perbaikan terhadap waktu untuk kasus terburuk pada algoritma Boyer-Moore. Namun, algoritma ini memerlukan *space* memori yang lebih banyak dalam impelementasinya.

III. METODE

A. Pencarian String Query dengan Algoritma Boyer-Moore

Pada pengaksesan basis data dengan menggunakan algoritma Boyer-Moore, penelusuran akan dilakukan untuk setiap elemen tabel yang diinginkan. Misalnya untuk pencarian nasabah bank dengan nama kota tempa tinggal adalah 'Bandung' akan digunakan algoritma seperti berikut:

```
Function searchQueryBM(nama_tabel, atribut,
pattern : string) → integer

Kamus

Procedure fillGoodSuffix(input
pattern:string)
{digunakan untuk mengisi tabel goodsuffix}

Procedure fillBadChar(input pattern:string)
{digunakan untuk mengisi tabel badchar}

m,n,j : integer

Algoritma

makeRightMostTable(pattern)
m ← pattern.length()
while i <= tabel<nama_tabel>.indeks do
   j ← 0
   y ← tabel<nama_tabel>.atribut[i]
   n ← y.length()
   while (j <= n-m) do
      for i=m-1 to x[i] = y[i+j]
         if (i<0) then
            j = goodsuffix[0]
         else
            j = max(goodsuffix[i],
            badchar[y[i+j]]-m+1+i)
         end if
      end for
   end while
end while
```

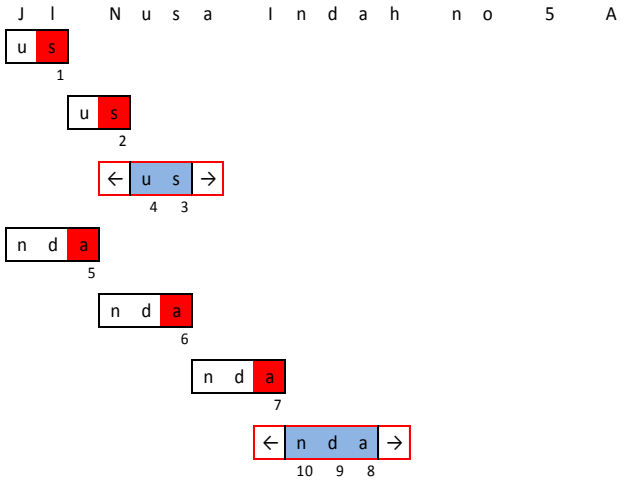
Dengan algoritma di atas, akan dilakukan pencarian dimulai dari elemen pertama dalam tabel yang diinginkan. Untuk setiap elemen tupel pada tabel akan diambil *string* dari atribut yang diinginkan sebagai teks yang akan digunakan untuk penelusuran. Langkah ini diulangi sampai semua tupel pada tabel tersebut sudah diperiksa. Setelah tupel terakhir diperiksa baru dikembalikan sebuah tabel yang berisi data nasabah pada indeks tabel hasil pencarian tersebut.

Untuk query dengan perintah **LIKE** yang akan dilakukan adalah dengan mencari setiap bagian *substring* yang dipisahkan oleh operator pembatas '%' atau '_' satu per satu sampai semua substring tersebut ditemukan lalu mengambil potongan kata tersebut.

Misalnya untuk pencarian dengan *query*:

```
SELECT *
FROM nasabah
WHERE     alamat_nasabah     like
'%us%nda%'
```

Pencarian pada alamat seorang nasabah dengan alamat 'Jalan Nusa Indah nomor 5 A' adalah sebagai berikut.



Gambar 6 Pencarian pattern LIKE dengan algoritma Boyer-Moore

Pada pencarian di atas terlihat bahwa pencarian akan dilakukan untuk setiap *substring* yang dipisahkan oleh tanda '%' dan kemudian menggeser indeks cakupannya sampai ditemukan spasi pada *string* teks yang ditelusuri tersebut. Dari gambar penelusuran di atas dapat dilihat bahwa dilakukan sepuluh kali pengecekan karakter sampai menghasilkan suatu jawaban.

B. Pencarian String Query dengan Algoritma Turbo-Boyer Moore

Pencarian *string* pada basis data dengan algoritma Turbo Boyer-Moore juga dilakukan untuk setiap elemen tabel yang diinginkan. Dengan kasus yang sama untuk algoritma Boyer-Moore, algoritma yang digunakan adalah sebagai berikut.

```
Function searchQueryBM(nama_tabel, atribut,
pattern : string) → integer
```

```
Kamus

Procedure fillGoodSuffix(input
pattern:string)
{digunakan untuk mengisi tabel goodsuffix}
```

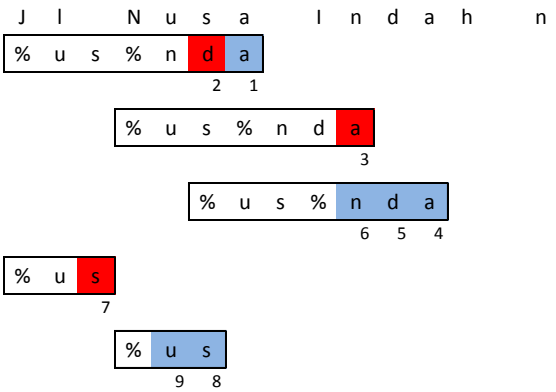
```
Procedure fillBadChar(input pattern:string)
{digunakan untuk mengisi tabel badchar}

m,n,j,shift,ts,bcs,k : integer
```

```
makeRightMostTable(pattern)
j ← u ← 0
m ← pattern.length()
shift ←m
while i <= tabel<nama_tabel>.indeks do
   j ← 0
   y ← tabel<nama_tabel>.atribut[i]
   n ← y.length()
   while (j <= n-m) do
      k = m-1
      while k>=0 and x[k]=y[k+j]
         k ← k-1
         if u≠0 and k=m-1-shift then
            k ← k-u;
         end if
      end while
      if i<0 then
         shift ← goodsuffix[0]
         u ← m - shift
      else
         v = m - 1 - k
         ts ← badchar[y[i+j]] - m + 1 + k
         shift ← max(ts,bcs)
         if(shift = goodsuffix[i]) then
            u = min(m-shift,v)
         else
            if(ts < bcs) then
               shift ← mac(shift,u+1)
            end if
            u ← o
         end if
      end if
   end while
end while
```

Proses pencarian *string* pada algoritma di atas juga sama seperti pada algoritma Boyer-Moore yaitu penelusuran dari tupel pertama pada tabel sampai pada tupel terakhir.

Untuk pencarian dengan perintah LIKE menggunakan algoritma Turbo Boyer-Moore, penelusuran dilakukan langsung untuk satu *string pattern* keseluruhan. Ilustrasi penelusuran dapat dilihat pada gambar berikut.



Gambar 7 Pencarian pattern LIKE dengan algoritma Turbo Boyer-Moore

Pada gambar di atas dapat dilihat bahwa penelusuran dilakukan tetap dari akhir *pattern* seperti Turbo Boyer-Moore seharusnya. Namun, ketika sudah menemukan *substring* terakhir pada *pattern*, *pattern* mulai dijalankan lagi dari awal. Dari gambar juga dapat dilihat bahwa pengecekan yang dilakukan adalah sembilan kali.

IV. Analisa

Dari kedua perbandingan alur penelusuran di atas dapat dilihat bahwa perbedaan kecepatan penelusuran *query* dengan algoritma Boyer-Moore tidak jauh berbeda dengan algoritma Turbo Boyer-Moore. Namun, karena kompleksitas waktu untuk kasus terburuk algoritma Boyer-Moore dapat dikurangi dengan menggunakan algoritma Turbo Boyer-Moore maka algoritma ini merupakan salah satu pilihan untuk memaksimalkan waktu pengaksesan basis data. Untuk perintah **LIKE** dalam query, penelusuran dengan algoritma Turbo Boyer-Moore melakukan lebih sedikit perbandingan karakter antara *pattern* dengan teks.

V. Simpulan

Dari perbandingan yang dilakukan pada makalah ini dapat diambill simpulan sebagai berikut:

1. Algoritma Boyer-Moore dapat digunakan untuk pengaksesan basis data dengan kecepatan yang baik.
2. Meski menghasilkan kecepatan yang hampir sama, algoritma Turbo Boyer-Moore dipilih untuk digunakan dalam pengaksesan basis data karena dapat mengurangi kompleksitas waktu kasus terburuk dari algoritma Boyer-Moore.
3. Tidak hanya untuk text editor, algoritma Boyer-Moore dan Turbo Boyer-Moore cocok digunakan untuk aplikasi manajemen basis data.

Referensi

[1] http://dev.mysql.com/doc/refman/5.5/en/index-btree-hash.html (Waktu akses : 6 Desember 2011)
[2] Munir, Rinaldi. 2009. Diktat Kuliah IF3051 Strategi Algoritma.Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung
[3] http://www-igm.univ-mlv.fr/~lecroq/string/node15.html#SECTION00150 (Waktu akses : 7 Desember 2011).
[4] http://www-igm.univ-mlv.fr/~lecroq/string/node14.html (Waktu akses : 7 Desember 2011)
[5] http://www.cs.cornell.edu/Courses/cs312/2002sp/lectures/lec25.htm (Waktu akses : 7 Desember 2011)
[6] http://www.w3schools.com/sql/sql_select.asp (Waktu akses : 8 Desember 2011)
[7] http://www.techonthenet.com/sql/like.php (Waktu akses : 8 Desember 2011)
[8] http://www.iti.fh-flensburg.de/lang/algorithmen/pattern/bmen.htm (Waktu akses : 8 Desember 2011)

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, 9 Desember 2010

Margaretha Siahaan
13509086