Penerapan Algoritma Mangkus dalam Fungsi Pencarian pada Database

Yusuf Ardi Nugroho (13510002)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13510002@std.stei.itb.ac.id

***Abstract*—Basis Data atau biasa disebut *Database* merupakan kumpulan informasi yang disimpan dalam suatu tempat. Dalam konteks pemrograman, *Database* adalah acuan yang dapat diakses oleh anggota sistem yang lain. Pada era sekarang, *Database* sudah sering digunakan, terutama dalam memudahkan penyimpanan dan pengaksesan informasi. Dengan posisi *database* sebagai *storage* yang menyimpan banyak informasi, muncul permasalahan baru yaitu ketika kecepatan akses yang melambat bila ukuran *database* sangat besar (ukuran data yang disimpan oleh *database*). Masalah kecepatan akses data ini sangat dipengaruhi oleh kemangkusan algoritma yang dipakai. Permasalahan inilah yang akan menjadi bahasan utama dalam makalah ini.**

***Index Terms* — Algoritma, *Database*, Mangkus, .**

I. PENDAHULUAN

Kehidupan manusia saat ini sangat dipengaruhi oleh aliran informasi yang ada. Berbagai teknologi telah diciptakan manusia untuk memudahkan aliran informasi baik dalam lingkup kecil maupun lingkup global. Hingga saat ini muncul berbagai fasilitas untuk memudahkan aliran informasi tersebut bergerak, sebut saja surat elektronik (surel atau *e-mail*), jejaring sosial (*social network*), *web*, *blog*, dan masih banyak contoh yang lainnya.

Dari contoh-contoh yang telah disebutkan di atas dapat dilihat bahwa arus informasi saat ini sangat bergantung dengan ***Internet*** dan juga **Teknik Pengolahan Data** yang digunakan di dalamnya. Makalah ini akan lebih membahas dari segi pengolahan data yang ada pada fasilitas-fasilitas tersebut.

Dari masa ke masa, teknik pengolahan data berkembang. Mulai dari pengolahan data manual (menggunakan kertas dalam tumpukan rekaman sebagai media), lalu ketika sistem pengolahan menggunakan sistem komputerisasi mulai digunakan sistem pengolahan berkas (kombinasi kertas secara komputerisasi), hingga sekarang menggunakan sistem manajemen informasi (menggunakan *database* yang terkomputerisasi).

Dari kesemua teknik pengolahan diatas ada beberapa poin yang akan selalu ada dan menjadi tolak ukur seberapa baik sistem tersebut. Teknik pengolahan data akan terdiri dari beberapa bagian: menyimpan, merubah, dan juga mengakses. Salah satu cara untuk mengolah data dalam sistem berbasis *internet* adalah dengan menggunakan *Database*. *Database* inilah yang akan menjadi tempat penyimpanan (*storage*), tempat terjadinya perubahan data, dan tempat tujuan akses data. Masalah-masalah yang biasanya muncul terkait teknik pengolahan data adalah masalah kapasitas (untuk menyimpan) dan masalah kecepatan akses (kecepatan akses juga berpengaruh pada bagian perubah)

Database yang ada sekarang mungkin jumlahnya sudah lebih sedikit dibanding masa sebelumnya namun memiliki kapasitas penyimpanan yang sangat besar. Sehingga seharusnya masalah dalam hal penyimpanan sudah terminimalisir.

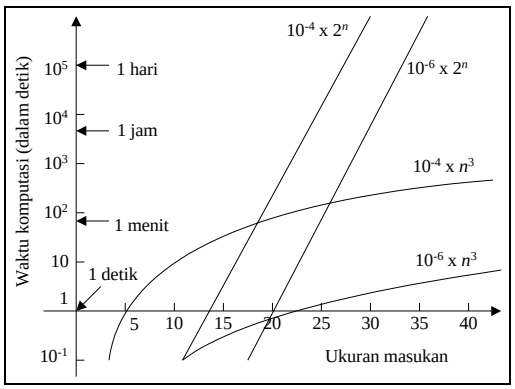
Masalah berikutnya adalah masalah kecepatan akses. Kecepatan pencarian suatu *keyword* ditentukan oleh kecepatan akses memori oleh sistem kepada database. Disini ada beberapa aspek yang mempengaruhi suatu data di *database* dapat diakses dengan cepat atau tidak. Aspek-aspek tersebut adalah bentuk database dan algoritma yang digunakan.

Bentuk algoritma dan struktur data yang baik akan dapat memercepat akses dari sistem ke database.

II. ALGORITMA YANG MANGKUS

Algoritma yang digunakan pada *database* akan menentukan langkah-langkah yang harus diambil oleh sistem dalam mengakses suatu memori (data). Contoh yang paling sering kita jumpai adalah masalah pencarian (*search*) dalam suatu *database*. Ketika kita mencari suatu *keyword* di dalam *database*, akan ada langkah-langkah yang dilakukan dalam mencari *keyword* tersebut. Seberapa mangkusnya algoritma yang dipakai akan menentukkan durasi yang diperlukan oleh sistem untuk mencari *keyword* tersebut.

Algoritma yang tidak mangkus akan membutuhkan waktu yang lama untuk mencari keyword tersebut, sedangkan algoritma yang mangkus mungkin membutuhkan waktu yang jauh lebih sedikit dibandingkan dengan algoritma yang tidak mangkus.



Gambar 2.1

Sebagai ilustrasi, pada Gambar 2.1 diperlihatkan perbedaan waktu eksekusi yang mungkin terjadi antara algoritma yang tidak mangkus dengan algoritma yang mangkus. Dampak dari algoritma yang mangkus akan sangat terasa ketika masukkan (jumlah data dalam *database*) berukuran sangat besar seperti *database* yang ada saat ini.

Berikut ini adalah contoh algoritma yang mangkus dibandingkan dengan algoritma tidak mangkus dalam konteks permasalahan yang sama.

Kode:

```
for($index=0;$index<= count($data);$index++) {
      $result = something_to_do();
}
```

Gambar 2.2 algoritma yang kurang mangkus

Kode:

```
$banyak_data = count($data);
for($index=0;$index<= $banyak_data;$index++) {
      $result = something_to_do();
}
```

Gambar 2.3 algoritma yang lebih mangkus

Kedua algoritma dalam Gambar 2.2 dan Gambar 2.3 adalah algoritma yang digunakan untuk mengeksekusi fungsi something_to_do() dalam studi kasus PHP. Pada algoritma dalam Gambar 2.2

Berikut ini beberapa Algoritma terkenal yang dapat mengatasi masalah pencarian string yang terdapat pada *Database*: Algoritma Rabin-Karp, Algoritma Knuth-Morris-Pratt, dan Algoritma Boyer-Moore.

II.1 Algoritma Rabin-Karp

Algoritma Rabin-Karp diperkenalkan pertama kali oleh Michael O. Rabin dan Richard M. Karp pada tahun 1987. Algoritma ini menggunakan tabel *array* dan metode *Hashing* dalam pengoperasiannya.

Metode *hashing* ini digunakan terutama untuk meningkatkan kecepatan pencarian dengan meningkatkan pengujian kesetaraan dalam teks.

Bagaimana hal tersebut bisa dilakukan? Pada faktanya, fungsi *hash* menyimpan bentuk *string* dalam bentuk lain yaitu enumerasi sehingga suatu *string* tertentu akan memiliki nilai enumerasinya sendiri-sendiri (unik). Karena suatu *string* hanya memiliki sebuah nilai enumerasi maka hal inilah yang digunakan oleh algoritma Rabin-Karp untuk mempercepat pencarian *string* dalam tabel *hash*. Dengan menggunakan metode seperti ini, akan terdapat kebocoran pada pencarian dalam teks yang panjang, karena pada teks yang panjang akan terjadi penomoran *string* yang sama meskipun *string* yang dituju berbeda. Sehingga dibutuhkan verifikasi lebih lanjut terhadap isi *string* tersebut. Hal ini sebenarnya dapat memakan waktu yang cukup lama apabila terjadi pada *substring* yang panjang. Namun fungsi *hash* yang baik akan menjamin kekurang seperti ini jarang terjadi, sehingga rata-rata waktu pencarian rata-rata menggunakan metode ini relatif baik.

Berikut ini adalah contoh pseudocode algoritma Rabn-Karp.

```
function RabinKarp(string s[1..n], string sub[1..m])
    hsub := hash(sub[1..m]);  hs := hash(s[1..m])
    for i from 1 to n-m+1
        if hs = hsub
            if s[i..i+m-1] = sub
                return i
        hs := hash(s[i+1..i+m])
    return not found
```

Gambar 2.1.1 *Pseudocode* algoritma Rabin-Karp

Nilai kompleksitas waktu terbaik algoritma Rabin-Karp adalah O(m) sedangkan untuk nilai kompleksitas terburuknya adalah O(mn).

Selain dari beberapa hal yang sudah dijelaskan diatas, terdapat sebuah keunggulan lain dari algoritma ini, yaitu bahwa algoritma ini merupakan algoritma yang paling mangkus dalam menghadapi persoalan *multi-set*.

II.2 Algoritma Knuth-Morris-Pratt

Algoritma Knuth-Morris-Pratt dikembangkan secara terpisah pada tahun 1966 oleh James H. Morris dan Vaughan R. Pratt serta Donald E. Knuth pada tahun 1967 tetapi diperkenalkan pada tahun 1977 secara bersama-sama oleh ketiga orang pengembangnya.

Algoritma ini meningkatkan kecepatan pencarian dengan mengembangkan algoritma *Brute-Force* dengan cara meningkatkan memori perbandingan yang telah dilakukan sehingga dapat melakukan pergeseran yang lebih besar.

Sistematika yang digunakan oleh algoritma ini adalah sebagai berikut:

1. Algoritma mencocokkan pattern pada awal teks.
2. Algoritma mulai mencari *string* yang bersesuaian dari kiri ke kanan, hingga mencapai salah satu kondisi berikut: karakter yang dibandingkan dengan *current pattern* tidak sesuai **atau** semua karakter yang berada di dalam *current pattern* cocok dan kemudian algoritma akan memberitahukan penemuan tersebut.
3. Algoritma menggeser *current pattern* berdasarkan tabel *next*, lalu algoritma akan mengulangi langkah

2 pada *current pattern* berikutnya sampai *current pattern* berada di akhir teks.

Contoh *Pseudocode* Algoritma Knuth-Morris-Pratt

```
procedure preKMP(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output kmpNext : array[0..n] of integer
)
Deklarasi:
i,j: integer

Algoritma
  i := 0;
  j := kmpNext[0] := -1;
  while (i < n) {
     while (j > -1 and not(P[i] = P[j]))
        j := kmpNext[j];
     i:= i+1;
     j:= j+1;
     if (P[i] = P[j])
        kmpNext[i] := kmpNext[j];
     else
        kmpNext[i] := j;
     endif
  endwhile
```

```
procedure KMPSearch(
    input m, n : integer,
    input P : array[0..n-1] of char,
    input T : array[0..m-1] of char,
    output ketemu : array[0..m-1] of boolean
)

Deklarasi:
i, j,next: integer
kmpNext : array[0..n] of interger

Algoritma:
preKMP(n, P, kmpNext)
i:=0
while (i<= m-n) do
    j:=0
    while (j < n and T[i+j] = P[j]) do
       j:=j+1
    endwhile
    if(j >= n) then
       ketemu[i]:=true;
    endif
    next:= j - kmpNext[j]
    i:= i+next
endwhile
```

Gambar 2.2.1 *Pseudocode* algoritma K-M-P

Nilai kompleksitas waktu algoritma ini adalah O(m+n) dengan m merupakan panjang teks dan n merupakan panjang pattern. Sedangkan kompleksitas ruang algoritma ini adalah O(n) pada memori internal jika teks dibaca pada file eksternal.

II.3 Algoritma Boyer-Moore

Algoritma Boyer-Moore diperkenalkan Robert S. Boyer dan J. Strother Moore pada tahun 1977. Algoritma ini merupakan salah satu algoritma paling mangkus dalam hal pencarian *string*. Hal ini dikarenakan sistem pencariannya yang berbeda dibandingkan dengan algoritma sejenis.

Algoritma Boyer-Moore menganalisis suatu *string* dengan mencocokkan suatu *string* dari sebelah kanan suatu *pattern*. Hal ini berbeda dengan kedua algoritma sebelumnya yang mencocokkan *string* dari sebelah kiri pattern. Hal ini pula yang memungkinkan penerimaan informasi yang lebih banyak dibanding menggunakan pencocokan dari sebelah kiri *pattern*.

Sistematika pencocokan *string* yang digunakan dalam algoritma ini adalah sebagai berikut:

1. Algoritma mulai mencocokkan *string* pada awal teks.
2. Algoritma mulai mencari *string* dari kanan ke kiri karakter-karakter yang ada pada *pattern* hingga salah satu kondisi berikut terpenuhi. Tidak ditemukannya karakter pada *pattern* dan teks yang dimaksud atau semua karakter yang berada di pattern cocok dan kemudian algoritma akan memberitahukan penemuan *string* ini.
3. Setelah langkah 2 selesai, algoritma akan menggeser *pattern* dan kemudian mengulangi kembali langkah 2 hingga *current pattern* berada di akhir teks.

Contoh *pseudocode* algoritma Boyer-Moore

```
procedure preBmBc(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output bmBc : array[0..n-1] of integer
)
Deklarasi:
  i: integer

Algoritma:
  for (i := 0 to ASIZE-1)
     bmBc[i] := m;
  endfor
  for (i := 0 to m - 2)
     bmBc[P[i]] := m - i - 1;
  endfor
```

```
procedure preSuffixes(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output suff : array[0..n-1] of integer
)

Deklarasi:
  f, g, i: integer

Algoritma:
  suff[n - 1] := n;
  g := n - 1;
  for (i := n - 2 downto 0) {
     if (i > g and (suff[i + n - 1 - f] < i - g))
        suff[i] := suff[i + n - 1 - f];
     else
        if (i < g)
           g := i;
        endif
        f := i;
        while (g >= 0 and P[g] = P[g + n - 1 - f])
           --g;
        endwhile
        suff[i] = f - g;
     endif
  endfor
```

```
procedure preBmGs(
    input P : array[0..n-1] of char,
    input n : integer,
    input/output bmBc : array[0..n-1] of integer
)
Deklarasi:
  i, j: integer
  suff: array [0..RuangAlpabet] of integer

  preSuffixes(x, n, suff);

  for (i := 0 to m-1)
     bmGs[i] := n
  endfor
  j := 0
  for (i := n - 1 downto 0)
     if (suff[i] = i + 1)
        for (j:=j to n - 2 - i)
           if (bmGs[j] = n)
              bmGs[j] := n - 1 - i
           endif
        endfor
     endif
  endfor
  for (i = 0 to n - 2)
     bmGs[n - 1 - suff[i]] := n - 1 - i;
  endfor
```

```
procedure BoyerMooreSearch(
    input m, n : integer,
    input P : array[0..n-1] of char,
    input T : array[0..m-1] of char,
    output ketemu : array[0..m-1] of boolean
)

Deklarasi:
i, j, shift, bmBcShift, bmGsShift: integer
BmBc : array[0..255] of interger
BmGs : array[0..n-1] of interger

Algoritma:
preBmBc(n, P, BmBc)
preBmGs(n, P, BmGs)
i:=0
while (i<= m-n) do
    j:=n-1
    while (j >=0 n and T[i+j] = P[j]) do
       j:=j-1
    endwhile
    if(j < 0) then
       ketemu[i]:=true;
    endif
    bmBcShift:= BmBc[chartoint(T[i+j])]-n+j+1
    bmGsShift:= BmGs[j]
    shift:= max(bmBcShift, bmGsShift)
    i:= i+shift
```

Gambar 2.3.1 *Pseudocode* Algoritma Boyer-Moore

Nilai kompleksitas dari algoritma ini adalah O(m,n), dan pada kasus terburuk akan melakukan 3n pencocokan karakter tetapi pada kasus terbaik hanya membutuhkan O(m/n) kali pencocokan karakter.

III. IMPLEMENTASI PADA DATABASE

Database yang ada sekarang merupakan kumpulan informasi yang disimpan dalam file eksklusif namun tetap dapat diakses secara *online*.

Berdasarkan data nilai kompleksitas yang terdapat pada ***bagian II*** maka dari ketiga algoritma tersebut yang paling cocok digunakan sebagai algoritma efisien pada database adalah algoritma Boyer-Moore. Meskipun kita tetap bisa menggunakan algoritma yang lainnya sesuai dengan jenis *database* yang akan kita buat.

IV. KESALAHAN YANG SERING TERJADI

Beberapa kesalahan yang biasanya dilakukan oleh pembuat database yang masih pemula adalah menggunakan tipe algoritma yang tidak mangkus, hal tersebut biasanya terjadi karena pembuatan database tersebut dilakukan terburu-buru dan kurang mempunyai visi pada pengembangan database tersebut kedepannya

Sehingga ketika kapasitas database meningkat, maka kecepatan akses yang diperoleh terus melambat dan mengganggu sistem tersebut.

V. KESIMPULAN

Bahasan yang terdapat dalam materi kuliah kompleksitas algoritma ternyata dapat memberi dampak yang sangat besar pada perkembangan kehidupan manusia secara global.

Diantaranya adalah dengan perkembangan teknologi pengolahan data, maka era aliran informasi secepat kilat (kecepatan tinggi) bukanlah hal yang mustahil terwujud.

REFERENCES

[1] Rinaldi Munir, *Matematika Diskrit edisi Ketiga*. Bandung : Informatika, 2005, ch 10.

[2] http://tegalcyber.org/showthread.php?tid=2492&highlight=struktur+data (dikunjungi pada tanggal 11 Desember 2011)

[3] http://id.wikipedia.org/wiki/Algoritma_Boyer-Moore (dikunjungi pada tanggal 11 Desember 2011).

[4] http://id.wikipedia.org/wiki/Algoritma_Knuth-Morris-Pratt (dikunjungi pada tanggal 11 Desember 2011).

[5] http://en.wikipedia.org/wiki/Rabin%E2%80%93Karp_algorithm (dikunjungi pada tanggal 11 Desember 2011).

[6] http://ninggarkusumawati.blogspot.com/2010/11/sejarah-data-base.html (dikunjungi pada tanggal 11 Desember 2011).

[7] http://ay-ayu.blogspot.com/2010/06/hashing.html (dikunjungi pada tanggal 11 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, 11 Desember 2011

Yusuf Ardi Nugroho (13510002)