Analisis Algoritma Knuth Morris Pratt dan Algoritma Boyer Moore dalam Proses Pencarian String

Rama Aulia

Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
e-mail: if16023@students.if.itb.ac.id

ABSTRAK

Makalah ini membahas tentang analisis algoritma pencocokan string. Pembahasan pada makalah ini lebih ditekankan pada pembahasan beberapa algoritma pencocokan string yang sering digunakan dalam implementasi pencarian suatu kata atau kalimat pada program aplikasi komputer.

Algoritma pencarian string merupakan suatu bagian penting dalam pemrosesan teks. Algoritma pencarian string merupakan komponen utama dalam implementasi software sistem operasi. Pencarian deret karakter tertentu dalam teks merupakan bahasan yang tidak akan pernah habis di dalam ilmu komputer.

Algoritma yang digunakan dalam proses pencocokan string antara lain Algoritma Brute Force, Algoritma Knuth-Morris-Pratt (KMP), Algoritma Boyer-Moore (BM), Algoritma Karp-Rabin, dan Algoritma Shift Or.

Karena algoritma untuk pencarian string atau kata di dalam teks banyak, maka hendaknya kita seharusnya mempertimbangkan kompleksitas algoritma yang ada dengan menganalisis kelebihan dan kekurangan dari algoritma pencarian string agar diperoleh efisiensi di dalam pencarian kata pada program aplikasi.

Kata kunci: *string matching algorithm, application program, Brute Force algorithm, Knuth Morris Pratt algorithm, Boyer Moore algorithm, Karp Rabin algorithm, Shift Or algorithm*

1. PENDAHULUAN

Pencarian string merupakan kegiatan yang sangat sering dilakukan oleh pengguna komputer. Dalam kehidupan sehari-hari, user (pengguna komputer) pasti berhubungan dengan yang namanya pencarian string. Untuk mencari suatu kata misalnya di dalam program Microsoft Word atau pun di web browser seperti Mozilla Firefox atau Internet Explorer, user akan berhubungan dengan bagian find yang merupakan penerapan langsung dari algoritma pencarian string di dalam program aplikasi.

Pencarian string di dalam teks disebut juga dengan pencocokan string (string matching atau pattern matching). Perumusan persoalan pencarian string yaitu dengan diberikannya teks atau long string dengan panjang n karakter dan pattern yaitu string yang akan dicari di dalam teks dengan panjang m karakter, dengan m lebih kecil dari n karakter.

2. ALGORITMA KNUTH MORRIS PRATT

2.1 Deskripsi Algoritma Knuth Morris Pratt

Algoritma Knuth Morris Pratt (KMP) dikembangkan oleh D. E. Knuth, bersama dengan J. H. Morris dan V. R. Pratt. Untuk pencarian string dengan algoritma Brute Force , setiap kali ditemukan ketidakcocokan dengan teks, maka pattern akan digeser satu karakter ke kanan. Berbeda dengan algoritma Brute Force, informasi yang digunakan masih dipelihara untuk melakukan jumlah pergeseran. Algoritma menggunakan informasi tersebut untuk membuat pergeseran yang lebih jauh, tidak hanya satu karakter

Perbandingan antara algoritma brute force dengan algoritma KMP ditunjukkkan dengan perpindahan posisi pattern terhadap posisi teks. Dimana dalam hal pencocokan string, pattern yang dicocokkan berawal dari kiri teks.

Kompleksitas algoritma pencocokan string dihitung dari jumlah operasi perbandingan yang dilakukan. Kompleksitas waktu terbaik dari algoritma brute force adalah $O(n)$. Kasus terbaik terjadi jika pada operasi perbandingan, setiap huruf pattern yang dicocokkan dengan awal dari teks adalah sama. Kompleksitas waktu terburuk dari brute force adalah $O(mn)$.

Jika dibandingkan dengan algoritma brute force, maka algoritma KMP mempunyai kompleksitas algoritma $O(m+n)$.

Definisi yang digunakan, Y adalah suatu alfabet dan

$z = z_1 z_2 \ldots z_k,\ k \in \mathbf{N}$,

adalah string dengan panjang k, yang dibentuk dari karakter di alfabet di dalam alfabet A. Awalan prefix dari z adalah upa-string w, dengan

$w = z_{k-b} z_{k-b+1} \ldots z_k$, $k \in \{1, 2, .., k-1\}$

Pinggiran dari z adalah substring r, sehingga

$r = z_1 z_2 \ldots z_{k,}$ dan $r = z_{k-b} z_{k-b+1} \ldots z_{k,}$ $k \in \{1, 2, .., k-1\}$

2.1.1 Fungsi Pinggiran

Algoritma KMP melakukan proses awal terhadap pattern dengan menghitung fungsi pinggiran Dengan adanya fungsi pinggiran ini, maka dapat dicegah pergeseran yang tidak berguna, seperti yang terjadi pada algoritma brute force.

Fungsi pinggiran hanya bergantung pada karakter yang ada di dalam pattern, tidak bergantung kepada karakter di dalam teks. Berikut adalah algoritma untuk mencari fungsi pinggiran di dalam pattern.

```
procedure HitungPinggiran(input m : integer,
P : array[1..m] of char, output b :
array[1..m] of integer)
{Menghitung nilai b[1..m] untuk pattern
P[1..m]}

Deklarasi
  k, i : integer

Algoritma
  b[1] ← 0
  q ← 2
  k ← 0
  for q ← 2 to m do
       while ((k>0) and (P[q] ≠ P[k+1])) do
         k ← b[k]
       endwhile
       if P[q]=P[k+1] then
         k ← k+1
       endif
       b[q]=k
  endfor
```

Adapun algoritma KMP yang lebih lengkap, sebagai berikut.

```
procedure KMPsearch (input m, n : integer,
input P : array[1..m] of char, input T :
array[1..n] of char, output idx : integer)
{Mencari kecocokan pattern P di dalam teks T
dengan algoritma Knuth-Morris-Pratt. Jika
ditemukan P di dalam T, lokasi awal kecocokan
disimpan di dalam peubah idx
Masukan : pattern P yang panjangnya m dan teks
T yang panjangnya n. Teks T direpresentasikan
sebagai string (array of character)
Keluaran : posisi awal kecocokan (idx). Jika
ditemukan, idx = -1
}

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

procedure HitungPinggiran(input m : integer,
P : array[1..m] of char, output b :
array[1..m] of integer)
{menghitung nilai b[1..m] untuk pattern
P[1..m]}

Algoritma
  HitungPinggiran(m, P, b)
  j ← 0
  i ← 1
  ketemu ← false
  while(i ≤ n and not ketemu) 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
       ketemu = true
     else
       i ← i + 1
     endif

  endwhile

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

3. ALGORITMA BOYER MOORE

3.1 Dekripsi Algoritma Boyer Moore

Algoritma pencarian string Boyer Moore merupakan algoritma pencarian string yang paling efisien dalam aplikasi sehari hari. Algoritma tersebut dikembangkan oleh Bob Boyer dan J Strother Moore pada tahun 1977.

Pada proses pencarian string, algoritma Boyer Moore membaca karakter-karakter dari pattern dari kanan ke kiri. Dalam kasus dimana jumlah karakter pada pattern lebih sedikit daripada jumlah karakter pada teks maka algoritma tersebut menggunakan 2 buah fungsi *precomputed*. 2 buah fungsi pengubah ini disebut *good-suffix shift***.**

Aturan pada *good-suffix shift* bertujuan untuk menangani kasus dimana terdapat pengulangan karakter pada pattern.

3.1.1 Prinsip Dasar

Algoritma Boyer Moore mempunyai empat konsep dasar di dalam proses pencarian string, yaitu :

1. *Preprocessing*
2. *Right-to-left-scan*
3. *Bad-character-rule*
4. *Good-suffix-rule*

Precomputation dari algoritma Boyer Moore terdiri dari bad-character preprocessing dan good-suffix preprocessing. Prinsip dasar yang pertama dari algoritma Boyer-Moore adalah melakukan perbandingan antara pattern yang dicari dengan teks. Perbandingan pattern dengan teks dilakukan dari arah kanan ke kiri. Perbandingan dimulai dengan membandingkan antara karakter paling kanan dari pattern dengan teks. Jika terjadi kecocokkan, maka perbandingan akan dilanjutkan dengan karakter yang di sebelah kiri dari yang dibandingkan sampai ke karakter pertama dari pattern. Jika terjadi ketidakcocokkan maka akan dilakukan pergeseran yang ditentukan oleh 2 fungsi pergeseran yaitu *bad character shift* dan *good suffix shift*. Aturan dari *bad character shift* dibutuhkan untuk menghindari pengulangan perbandingan yang gagal dari suatu karakter dalam teks dengan pattern. Aturan dari *good suffix shift* dibutuhkan untuk menangani kasus yang di dalamnya terdapat pengulangan karaker pada pattern.

Penjelasan algoritma Boyer Moore sebagai berikut. Anggap sebuah string

$B = B[1], \ldots, B[i]$

dan diperiksa string lain

$M = M[1], \ldots, M[j],\ j \leq i$

apakah string M merupakan substring dari string B atau terdapat substring

$B' = B[k], \ldots, B[k+j],\ k + j \leq i$

dengan hubungan bahwa $B' \equiv M$. Algoritma Boyer – Moore adalah algoritma yang memanfaatkan fakta bahwa:

1. Jika terdapat setidaknya satu elemen dalam $\{s \mid s \in B'\}$ yang tidak berada dalam T maka

   $B' \neq M$

   jika kondisi tersebut yang didapat, maka lakukan perubahan nilai

   $k \leftarrow k + j$

   selama $k + j \leq i$, sehingga B' diganti seluruhnya

2. Jika terdapat simbol $a = B[k + l]$ dan $a' = M[m]$ dimana $a = a'$, namun $l \neq m$ terdapat kemungkinan ada k' dan substring lain dari B,

   $B'' = B[k'], \ldots, B[k' + j],\ k' + j \leq i$

   dimana

   $B'' \equiv M$

   Masih terdapat masalah bila hanya kedua langkah tersebut yang digunakan, karena bila string yang kita hadapi misalnya adalah

   abbabbababbabababbbaaaa

   dan string yang ingin dicari yaitu

   aaaa

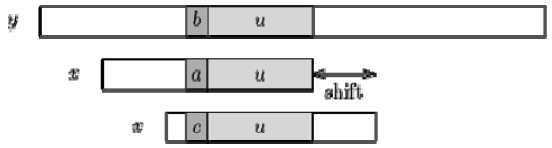
   lalu pencocokan linear dilakukan dari indeks terkecil, maka pergeseran indeks k akan terlalu sering dilakukan; batas bawah unjuk kerja pencarian dengan cara ini mendekati unjuk kerja algoritma *brute-force*. Langkah optimasi lebih lanjut adalah memulai pencocokan dari elemen dengan indeks terbesar ($S[k + j]$ dan $T[j]$), yang berakibat meminimalkan pergeseran indeks k.

Untuk asumsi dimana terjadi *mismatch* antara karakter $x[i] = a$ pada pattern dengan karakter $y = [i+j]$ pada teks. Maka akan menghasilkan

$x[i+1 .. m-1]=y[i+j+1 .. j+m-1]$=u dan $x[i] \neq y[i+j]$

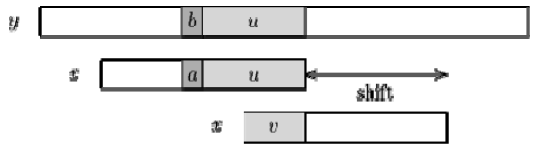
Good suffix shift terdapat pada segmen

$y[i+j+1 .. j+m-1]=x[i+1 .. m-1]$



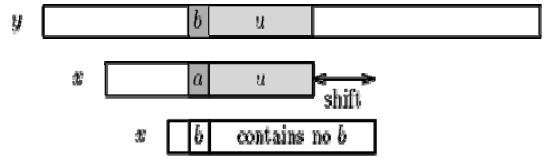
Gambar 1. ***good suffix shift***

Jika tidak terdapat segmen, maka pergerakan terjadi dalam perbandingan dari akhiran v pada $y[i+j+1 .. j+m-1]$

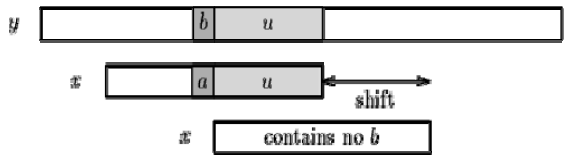


Gambar 2. ***Good suffix shift,*** **hanya akhiran** ***u*** **terdapat kembali pada** ***x***

Bad character shift terjadi pada perbandingan karaker teks $y[i+j]$



Gambar 3. ***bad character shift, a*** **terdapat pada** ***x***



Gambar 4. ***bad character shift,*** **b tidak terdapat pada** ***x***

Analisis kompleksitas algoritma Boyer Moore menyangkut 3 hal yaitu komplekstitas waktu, kompleksitas ruang, dan waktu pemrosesan. Kasus terbaik pada algoritma Boyer Moore terjadi jika pertama kali masing-masing percobaan membandingkan simbol teks tidak cocok dengan yang ada di pattern. Sehingga kompleksitas waktu yang terbaik dengan notasi big O yaitu O(N/M). Kompleksitas waktu untuk kasus terburuk adalah O(Nm). Pada umumnya, kompleksitas waktu yang dibutuhkan untuk kasus rata-rata adalah O(N/M).

Waktu yang dibutuhkan oleh algoritma Boyer Moore dalam preprocessing fungsi *good suffix shift* adalah O(M). Sedangkan untuk kompleksitas preprocessing aturan *bad character* adalah O(M + |*alphabet*|) untuk kompleksitas ruang dari algoritma Boyer Moore adalah O(m + |*alphabet*|).

Untuk perbandingan dengan algoritma lain seperti algoritma Brute force. Kelebihan algoritma Boyer Moore terdapat pada kecepatan untuk pattern yang panjang. Selain itu, algoritma Boyer Moore lebih cepat daripada algoritma Brute Force. Sedangkan untuk kekurangan yang dimiliki oleh algoritma Boyer Moore yaitu antara lain tidak bagus untuk *binary string* dan lebih lambat untuk pattern yang pendek.

4. KESIMPULAN

Proses pencarian string merupakan hal yang sangat fundamental di dalam program aplikasi karena mencari kata di dalam suatu teks merupakan hal yang sangat sering dilakukan. Algoritma pencarian string jenisnya bermacam – macam. Untuk algoritma Knuth Morris Pratt, pergeseran pattern dilakukan tidak hanya satu karakter seperti pada Algoritma Brute Force, hal tersebut bergantung pada informasi yang disimpan untuk pergeseran. Untuk Algoritma Boyer Moore, terdapat keunikan dimana perbandingan pattern dilakukan mulai dari kanan ke kiri.

REFERENSI

[1] Munir, Rinaldi. (2007). Diktat Kuliah IF2251 Strategi Algoritmik. Program Studi Teknik Informatika, Institut Teknologi Bandung.

[2] Boyer R.S., Moore J.S., 1977, A fast string searching algorithm. *Communications of the ACM*. 20:762-772.

[3] *http://en.wikipedia.org/wiki/Boyer-Moore_string_search_algorithm.html*

[4] *http://en.wikipedia.org/wiki/Knuth_morris_pratt_algorithm.html*