Pengembangan Algoritma Boyer Moore

Aldyaka Mushofan - 13512094[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]aldyaka.mushofan@s.itb.ac.id

***Abstract*—Pencarian dan pencocokan pola merupakan salah satu permasalahan dalam ilmu komputer yang paling sering dipelajari. Dalam memecahkan permasalahan ini, banyak digunakan berbagai solusi dan salah satunya ialah dengan menggunakan algoritma Boyer-Moore. Pada algoritma ini pun, masih sering dipelajari dan dikembangkan oleh para ahli sehingga menghasilkan berbagai macam versi dari algoritma Boyer-Moore yang tidak menutup kemungkinan pula akan dihasilkan versi-versi lain dari algoritma ini.**

***Index Terms*—Boyer-Moore, Algoritma pencarian dan pencocokan pola, Kompleksitas waktu, .**

I. PEMBUKAAN

Pencarian dan pencocokan pola merupakan salah satu permasalahan di ilmu komputer yang paling banyak dipelajari. Untuk melakukan pencarian dan pencocokan string dalam komputer, digunakan algoritma tertentu. Algoritma pencarian dan pencocokan pola bertujuan untuk mencari pola yang diberikan apakah ada atau tidak dalam suatu teks atau data lainnya.

Algoritma pencarian dan pencocokan pola ini dalam perkembangannya, sampai saat ini dihasilkan berbagai macam algoritma, antara lain: Algoritma *Brute-force*, Algoritma *Knuth-Morris-Pratt(KMP)*, *Finite Automata*, Algoritma *Rabin-Karp*, *Longest Common Subsequence(LCS)*, dan Algoritma *Boyer-Moore(BM)*.

Masing-masing dari algoritma di atas memiliki kelebihan dan kelemahan masing-masing. Beberapa ahli di ilmu komputer melakukan berbagai penelitian dan percobaan terhadap beberapa algoritma yang telah ada untuk meningkatkan performa dari algoritma-algoritma tersebut, namun ada juga algoritma yang tidak begitu diminati untuk dikembangkan lagi oleh para ahli dan para pengembang, mereka memilih untuk mengembangkan algoritma pencarian lainnya yang telah memiliki tingkat kinerja yang sudah relatif bagus. Pada tulisan ini, pembahasan seputar algoritma solusi yang digunakan untuk mengatasi permasalahan pencarian dan pencocokan pola/string akan difokuskan terhadap Algoritma Boyer-Moore dan pengembangan dari Algoritma Boyer-Moore.

II. PENGEMBANGAN ALGORITMA BOYER-MOORE

Pada mulanya, Algoritma Boyer-Moore ini diajukan oleh 2 ahli, yakni Robert S. Boyer dari Stanford Research Institute dan J Strother Moore dari Xerox Palo Alto Research Center pada tahun 1977. Nama dari algoritma ini kemudian diambil dari kedua nama ahli yang telah mengajukan algoritma ini di jurnalnya[1].

Algoritma ini berbeda dengan algoritma yang telah dikembangkan sebelumnya. Pada algoritma lainnya, pencarian dan pencocokan dilakukan dilakukan dari kiri ke kanan, sedangkan pada Algoritma Boyer-Moore, pencarian dan pencocokan dilakukan dari kanan ke kiri, disesuaikan dengan panjang pola dan panjang teks yang akan dicari pola di dalamnya. Sebagai contoh, bila kita akan melakukan pencarian pola “abcd”, maka akan dilakukan pencarian dari ‘d’ terlebih dahulu, bila sesuai, maka akan dilanjutkan ke ‘c’ dan seterusnya hingga semua sesuai atau hingga ditemukan ketidakcocokan.

Dalam proses pencarian dan pencocokan pola, algoritma ini akan membuat dua buah tabel. Tabel pertama menjelaskan heuristik yang sesuai dan tabel kedua menjelaskan tentang he uristik kemunculan. Dalam proses pencarian dan pencocokan pola, bila ditemukan ketidakcocokan antara pola dengan teks sumber, maka tabel pertama akan memberitahukan berapa banyak karakter yang akan dilompati (tidak dilakukan pengecekan terhadapnya). Untuk lebih jelasnya, perhatikan contoh berikut:

Misalkan pola yang ingin kita cari ialah “EDF” dan kita ingin mencari pola tersebut pada teks “ABDCAEDFE”.

Pola: EDF
Kalimat: ABDCAEDFE

Pencocokan pertama:

```
ABDCAEDFE
EDF
```

Pencocokan dimulai dari F dan ketika mencocokkan F dengan karakter pada kalimat ditemui ketidakcocokan, dan ditemui bahwa D berada di posisi ketiga pada kalimat, maka pola digeser sebanyak 1 langkah ke kanan (selisih dari posisi D yang ada di pola).

```
ABDCAEDFE
 EDF
```

Ketika mencocokkan F, ternyata yang ditemui ialah C,

dan C tidak ditemukan di pola, maka penggeseran dilakukan sebesar panjang dari pola (sebesar 3 langkah).

ABDCAEDFE
EDF

Ditemukan lagi bahwa F tidak cocok dengan karakter yang satu posisi dengan F pada kalimat. Selain itu, posisi D pada kalimat selisih satu dengan posisi D pada pola, sehingga pola bergeser 1 langkah ke kanan.

ABDCAEDFE
EDF

Pertama, mencoba mencocokkan F dengan karakter pada kalimat, karena sesuai, maka pencocokan dilanjutkan ke D, dan sekali lagi cocok, sehingga dilanjutkan ke karakter paling awal pada pola dan ditemukan bahwa cocok, sehingga akan dikembalikan bahwa pola tersebut ada di kalimat tersebut pada posisi ke 6.

Algoritma Boyer-Moore biasanya akan bekerja sangat bagus bila dipakai untuk pola yang relatif panjang, sedangkan untuk pola yang relatif pendek, kompleksitasnya mencapai O(n). Algoritma ini dinilai sudah cepat untuk menangani pencarian dan pencocokan terhadap pola dalam berbagai kasus. Namun, beberapa ahli masih melakukan penelitian dan percobaan terhadap Algoritma Boyer-Moore ini. Setelah beberapa penelitian dan percobaan terhadap Algoritma Boyer-Moore, didapat beberapa cara yang dapat meningkatkan performa pencarian dan pencocokan terhadap pola oleh algoritma ini. Hasil dari pengembangan para ahli ini antara lain: Penggabungan Boyer-Moore dengan *Reverse Factor*, Algoritma BM', Algoritma Boyer-Moore-Horspool dan Algoritma Boyer-Moore-Galil.

III. PENGGABUNGAN BOYER-MOORE DENGAN *REVERSE FACTOR*

Pengembangan ini diusulkan oleh M. Crochemore, A. Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski dan W. Rytter pada tahun 1994. Pada solusi ini, terdapat penggabungan algoritma pencarian dan pencocokan string yang satu di antaranya ialah pengembangan dari algoritma satunya. Algoritma *Reverse Factor* ialah pengembangan dari algoritma Boyer-Moore. Pada kasus rata-rata, performa algoritma *Reverse Factor* lebih baik dibanding algoritma Boyer-Moore, baik untuk kasus pola yang pendek maupun untuk kasus pola yang panjang. Algoritma *Reverse Factor* dengan pola yang memiliki panjang minimal 2 karakter akan memiliki kompleksitas rata-rata sebesar O(n log(m)/m).

Sebelum memasuki perbandingan algoritma BM yang digabungkan dengan algoritma *Reverse Factor*, kita akan membahas tentang algoritma *Reverse Factor*. Algoritma RF ini ialah pengembangan dari algoritma BM yang menggunakan sufiks terkecil dari pola yang telah dibalik. Sufiks terkecil dari suatu kata *w*ialah DFA yang bisa ditulis sebagai berikut:

$$\mathbf{S(w) = (Q,q_0,T,E)}$$

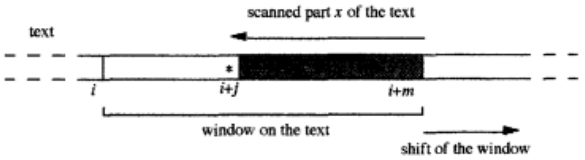
Bahasa yang diterima oleh S(w) adalah:

$\mathbf{L(S(w))}$ = {u di $\mathbf{\Sigma^*}$: terdapat v di $\mathbf{\Sigma^*}$ dimana w=vu}

Pada algoritma RF, kemungkinan kasus terburuk baru dapat ditangani dalam kompleksitas waktu *quadratic*, akan tetapi nilai rata-rata kasusnya cukup optimal, mencapai kompleksitas waktu $O(n \log_\sigma (m)/m)$.

Salah satu perbedaan dari algoritma *Reverse Factor* dengan algoritma Boyer-Moore ialah proses pencocokan. Pada algoritma Boyer-Moore, pencocokan dilakukan sepanjang segmen, tetapi Algoritma RF mencocokkan kata dengan faktor-faktor dari pola yang ada. Selanjutnya, maka pola akan digeser ke kanan, dan melupakan hasil pencocokan sebelumnya dan memulai ulang pencarian dan pencocokan.

Metode yang digunakan untuk meningkatkan performa dari algoritma BM ialah metode simulasi dinamis dari algoritma BKR. Pengubahan algoritma BM pada solusi ini mampu meningkatkan performa pada batas atas yang senilai 3n. Struktur umum dari penggabungan algoritma BM dan RF dapat digambarkan sebagai berikut:



Gambar 1 Ilustrasi Penggabungan Algoritma BM dengan RF

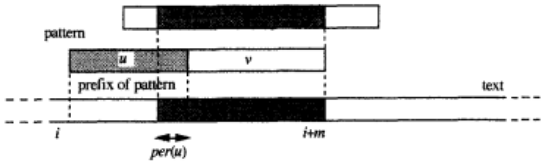
Sementara itu, algoritma ini bila dituliskan dalam notasi algoritmik dapat ditulis sebagai berikut:

Algorithm BM-RF

```
i=0;
while I <= n – m do
{
    Samakan posisi pola t[i+1….i+m] dengan teks;
    Scan teks dari kanan ke kiri dari posisi m+i;
     x adalah bagian dari teks yang dibaca;
    if x = p then sama di posisi I;
    hitung penggeseran;
    i = i + penggeseran;
```

}**end**

Untuk mempercepat algoritma RF, kita harus mengingat prefiks *u* dari ukuran penggeseran m dari pola. Selanjutnya, seperti pada algoritma BM biasa, pembacaan pola dari awal teks hingga bagian *v* dari pola dilakukan dari kanan ke kiri. Jika pembacaan dari batas *u* hingga *v* menghasilkan nilai benar (cocok semua), maka kita telah mencapai titik putusan. Pada titik ini, kita melakukan pembacaan ulang dari bagian *u* dibanding melakukan pembacaan hingga *u* ditemukan tidak cocok.



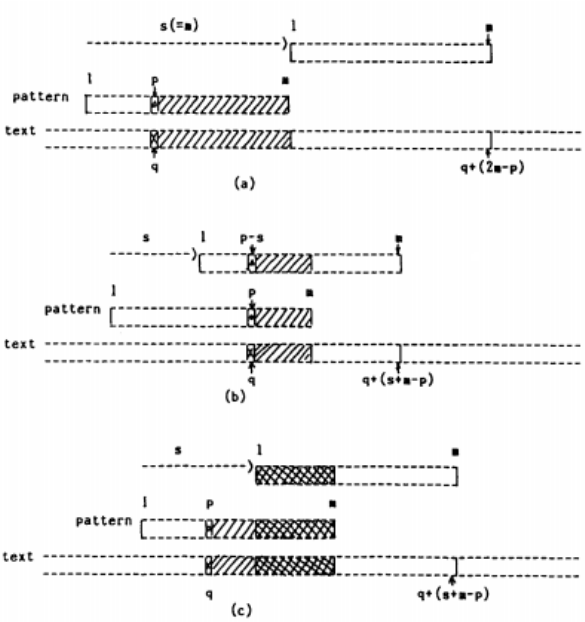
Gambar 2 Mempercepat Algoritma Reverse Factor

IV. ALGORITMA BM'

Algoritma ini dikembangkan oleh Zhu Rui Feng dan Tadao Takaoka yang sama-sama dari Jepang. Ide pokok dari algoritma ini ialah mengubah perhitungan awal yang berguna untuk mempersiapkan tabel yang akan digunakan. Pada algoritma BM biasa, pembacaan awal dilakukan terhadap satu karakter saja, sedangkan pada algoritma BM', pembacaan awal dilakukan terhadap dua karakter sekaligus.

Misalkan pada algoritma Boyer-Moore fungsi penggeseran pola dilambangkan oleh *D* dan *DD'*. Fungsi *DD'* melakukan penggeseran pola ke kanan harus sesuai dengan semua karakter sebelumnya dan membawa karakter baru yang berbeda dengan yang ada di teks. Fungsi *DD'* dapat dituliskan sebagai berikut:

DD'[p] = min {s+m-p|(s>=1) dan (s>=p atau
pola [i-s] = pola [i], p < i <= m)}



Gambar 3Ilustrasi Fungsi DD'

Fungsi *D* melakukan penggeseran sehingga karakter yang sama pada teks dan pola sejajar. Fungsi *D* dapat dituliskan sebagai berikut:

D[ch] = min{s|s=m atau (0<s<m dan pola[m-s] = ch)}

Dimana ch adalah karakter pada pola yang sesuai dengan sebuah karakter yang ada pada teks.

Pada Algoritma BM', terdapat perbedaan pada fungsi tersebut. Pada Algoritma BM' fungsi D diubah sedemikian rupa sehingga dapat mengoperasikan dua karakter sekaligus. Fungsi D pada algoritma ini bisa ditulis sebagai berikut:

D[ch1,ch2] = min {s|(s=m) atau
(s=m-1 dan ch2 = pola[1]) atau
(0<s<m-1 dan pola [m-s-1]=ch1
Dan pola[m-s] = ch2)}

ch1 dan ch2 ialah karakter dari pola yang ingin dicari

V. ALGORITMA BOYER-MOORE-HORSPOOL

Algoritma Boyer-Moore-Horspool ini dikembangkang oleh Timo Raita dari University of Turku, Finlandia. Perbedaan yang ada dari Algoritma BM dengan Algoritma BMH ini ialah tabel yang digunakan. Pada algoritma BM, tabel yang digunakan ialah tabel yang heuristik yang sesuai (match heuristic) dan heuristik kemunculan (occurrence heuristic), namun pada algoritma Boyer-Moore-Horspool (BMH), tidak semua tabel yang digunakan, hanya tabel *occurrence heuristc* saja. Karena tidak memakai tabel *match heuristic*, maka peloncatan—peloncatan karakter memiliki aturan yang berbeda. Pada setiap karakter pada teks yang terdapat pada pola, maka langsung akan disejajarkan dengan karakter yang ada pada pola dan dilihat apakah karakter sisa sesuai atau tidak.

Pencarian dan pencocokan pola yang dilakukan tidak memperhatikan dimana posisi yang terjadi ketidakcocokan. Dengan menggunakan algoritma ini, kasus terburuknya ialah dimana setiap karakter pada teks terdapat pada pola, tetapi tidak ditemukan kesesuaian dari karakter dan pola. Contohnya:

Pola:

ACDE

Teks :

ACDACCDECDACDCDECDE

Maka secara teori akan dilakukan pengecekan di tiap karakter pada teks tersebut.

Kerugian dari solusi ini ialah melakukan pengecekan karakter yang bukan berada di ujung 2 kali. Namun, hal ini bisa diatasi dengan konsekuensi program dapat berjalan lebih lambat dibanding program semula. Program akan berjalan lebih lambat karena logika yang digunakan semakin rumit. Selain kerugian tersebut, keuntungan pemakaian solusi ini ialah kita tidak mengetahui tentang karakter yang tidak berada dalam pola.

Dalam pengujiannya, alogritma Boyer-Moore-Horspool dapat bekerja dengan cepat untuk teks acak.

Bila ditulis dalam notasi algoritmik, Algoritma Boyer-Moore-Horspool dapat ditulis sebagai berikut:

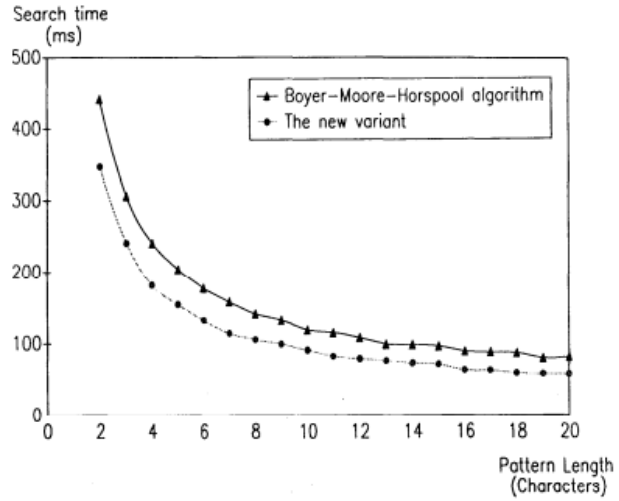
```
procedure bmhsearch(teks: string; n: integer; pola: string; m: integer);

kamus:
        i, j, k: integer;
        ch:  karakter;
        occtype: array [chr(0)..chr(panjang-1)] of integer

begin
/* mempersiapkan tabel */
for ch:= chr(0) to chr(panjang-1) do
        occheur[ch]:=m;
for j:= 1 to m-1 do
        occheur[pola[j]]:=m-1;

/* menambahkan elemen sentinel pada ujung teks dan pola */
pola[0] := sentinel;
teks[0] := sentinel;

/* pencarian  dan pencocokan*/
i:=m;
while i<=n do
        begin
          k:=i;
          j:=m;
          while teks[k] = pola[j] do
                  begin
                    k:= k-1;
                    j:=j-1;
                  end;
          if j =0 then
                writeln(output, "Posisi yang sesuai: ", k+1);
          i := i+occheur[teks[i]];
        end;
end;
```

Selain versi di atas, dikembangkan juga versi yang menggunakan *inner loop*(pengulangan di bagian dalam). Dalam notasi algoritma, pengulangan tersebut dapat ditulis sebagai berikut:

```
begin
        k : = i – 1 ;
        for j : = mminusone downto 2 do
        begin
          if txt[k] <> pat[j] then goto 1;
          k : = k – 1 ;
        end;
        writeln(output,  ¢ Match
end;
1:
i : = i + occheur[txt[i]];
```

Pada pengujian algoritma ini, Timo Raita menggunakan teks sepanjang 29.550 karakter. Alfabet menggunakan ASCII dengan ukuran alfabet secara teori 128 dan ukuran aktualnya 85. Pola yang digunakan ialah pola hasil pengacakan dari teks dengan panjang 2-20.

Pencarian dilakukan sebanyak 30 kali tiap masing-masing panjang pola. Pengujian ini dilakukan menggunakan Micro Vax-11 microcomputer menggunakan Pascal



Gambar 4 Grafik Performa Pencarian dan Pencocokan Pola dengan Algoritma BMH dan Variasi Barunya

Dari grafik di atas, didapat bahwa variansi baru dari BMH mampu meningkatkan performa dari BMH sebesar 21-27 persen. Karena menggunakan teknik seleksi pola, didapat minimal 1 kali pola yang cocok.

VI. ALGORITMA BOYER-MOORE-GALIL

Algoritma Boyer-Moore-Galil ini diajukan oleh Zvi Galil. Zvi Galil memodifikasi algoritma BM sehingga

pada pengujian menggunakan hasil dari Guibas dan Odlyzko dihasilkan 14n waktu yang diperlukan dalam kasus terburuk pada permasalahan pencarian dan pencocokan pola.

Pada Oktober 1985, Alberto Apostolicao dan Raffaele Giancarlo pada jurnal mereka mengajukan pengembangan dari algoritma ini. Hasil dari algoritma ini didapat beberapa hal yang menarik:

1. Performa bekerja maksimal pada pencarian dan pencocokan pola yang memiliki panjang karakter 2n-m+1
2. Pembuktian linear sangat sederhana
3. Fungsi *DD* bisa digunakan menggantikan *DD'* tanpa mempengaruhi hasil.

Algoritma Boyer-Moore-Galil yang diajukan oleh Zvi Galil dapat dituliskan dapat notasi algoritmik sebagai berikut:

```
procedure bmgsearch
j:=m;
l:=0;
do while j<=n
      begin
        do i:= m to l by -1 until pola[i] != teks[j-m+i]
        if i=l then
              begin
                output(sama di j-m+i);
                l:=l0;
                l := 0;
              end;
        else
              l:=0;
              j:=j+s[teks[j-m+i]r,i]
      end
```

Dari algoritma di atas, pada kasus terburuk hanya diperlukan waktu linier untuk menyelesaikannya.

Sedangkan algoritma dari pengembangan algoritma Boyer-Moore-Galil yang dikembangkan oleh Alberto Apostolicao dan Raffaele Giancarlo dapat ditulis sebagai berikut:

```
procedure bmg2
j:=m;
do while j<=n
      begin
        do i:=m to 0 by-max(1,skip[j-m+i])
              until Q[i,skip[j-m+i]] atau ((skip[j-m+i]
              = 0)
              andif (pola[i] = teks[j-m+i]))
       if i<=0 then
        begin
              output (sama di j-m+1);
              i:=0
        end
        skip[j]:=m-I; j:=j+s'[teks[j-m+i],i]
      end
```

Algoritma ini memiliki cara kerja yang mirip dengan algoritm sebelumnya setelah menemukan adanya pola yang berulang.

Pada algoritma Boyer-Moore-Galil yang sudah dikembangkan ini dapat dijelaskan secara teori dan pembuktiannya, yakni sebagai berikut:

Teori 1: Algoritma BMG2 ini mendeteksi semua kejadian dari pola pada teks dengan kemampuan optimal pada pencarian dan pencocokan pola dengan panjang karakter 2n-m+1

Pembuktian: Semua kemunculan dari pola pada teks didata. Pemakaian **andif** pada algoritma mengakibatkan algoritma ini tidak melakukan pengecekan berulang pada kondisi yag sudah diketahui salah. Setiap pencarian dan pencocokan terhadap karakter pada pola dan teks memiliki kemungkinan terjadi ketidakcocokan atau kecocokan. Jika terjadi kecocokan, maka karakter pada teks tersebut akan dilewati terlebih dahulu. Dengan demikian, bisa dilihat bahwa jumlah ketidakcocokan tidak akan melebihi n-m+1. Karena disaat terjadi ketidakcocokan akan dilakukan penggeseran terhadap pola, maka jumlah karakter yang menghasilkan kinerja maksimal ialah dengan panjang 2n-m+1.

Teori 2: Prosedur prefiks secara tepat menghitung prefix[1:m]

Pembuktian: *prefix*[1:d_1-1] akan diisi dengan 0 pada saat inisiliasasi. Asumsikan sekarang pada *prefix* penghitungan dilakukan dengan tepat pada posisi i dimana w[i+1] = w[1]. *Prefix*[i] sama dengan suatu integer p>=1. Misalkan d merupakan integer terkecil dimana w[i+d+1] = w[1]. Misal j = i+d. Pengulangan akan diperlukan untuk menghitung *prefix*[i] pada kasus k-d_f <=0. Bisa ditunjukkan bahwa untuk kasus k-d_f > 0 dapat ditangani dengan konsisten. Kasus akan terpecah menjadi 2 bagian, kedua sub kasus tersebut akan sama-sama mengeksploitasi posisi j dengan replika dari prefiks yang dimulai dari i. Nilai dari *prefix*[i] harus digandakan dari $prefix_w$[d-1] jika panjang kata kurang dari k-d_f. Jika tidak, *prefix*[i] memiliki nilai panjang minimal

sebesar k-d_f dan kita hanya perlu melakukan pencarian dan pencocokan terhadap karakter untuk memperpanjangnya juga.

Teori 3: Prosedur *Prefix* menghabiskan waktu O(m)

Pembuktian: Jumlah kerja untuk melakukan pengaksesan posisi I dimana w[i] = w[1] dibatasi oleh m. Tiap kerja yang berkaitan dengan pencarian dan pencocokan pola w[k] dan w[i+k+1] bisa kita beri "tanggung jawab" untuk melakukan satu tugas pencarian dan pencocokan pola pada posisi i+k+1. Tiap posisi tidak bisa diberikan lebih dari satu tugas pencarian dan pencocokan pola. Ketidakcocokan antara pola dengan teks tidak dapat melebihi p<=m, yang merupakan kesimpulan dari pembuktian ini.

VII. KESIMPULAN

Saat ini, telah banyak pengembangan algoritma yang bisa mengatasi permasalahan dari pencarian dan pencocokan pola yang merupakan salah satu permasalahan yang paling sering dipelajari di ilmu komputer. Tiap algoritma pengembangan dari algoritma Boyer-Moore memiliki keunggulan dan kelemahan masing-masing. Pemakaian algoritma yang tepat sesuai keunggulannya dapat meningkatkan kinerja dari program tersebut, sehingga penting bagi kita untuk mengetahui kelemahan dan keunggulan masing-masing algoritma dan penting juga untuk mengetahui bagaimana kasus yang akan kita hadapi, sehingga tidak salah memilih solusi algoritma yang akan kita terapkan.

REFERENCES

[1] Robert S. Boyer, J Strother Moore, "A Fast String Searching Algotihm", Oktober 1977.

[2] M. Crochemore, A.Czumaj, L. Gasieniec, S. Jarominek, T. Lecroq, W. Plandowski dan W. Rytter,"Speeding Up Two String-Matching Algorithms", Desember 1994

[3] Zhu Rui Feng, Tadao Takaoka, "On Improving the Average Case of The Boyer-Moore String Matching Algorithm", 1987

[4] Timo Raita, "Tuning the Boyer–Moore–Horspool String Searching Algorithm", Oktober 1992.

[5] Alberto Apostolico, Raffaele Giancarlo,"The Boyer-Moore-Galil String Searching Strategies Revisited", Oktober 1985

[6] http://www-igm.univ-mlv.fr/~lecroq/string/node23.html

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, 18 Mei 2014

Aldyaka Mushofan - 13512094