Perbandingan Penggunaan Algoritma BM dan Algoritma Horspool pada Pencarian *String* dalam Bahasa Medis

Evlyn Dwi Tambun / 13509084
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13509084@std.stei.itb.ac.id

***Abstrak*—Perkembangan pengetahuan di dunia medis dewasa ini sangat cepat sehingga literatur dan data medis lainnya juga ikut bertambah dengan sangat cepat. Seringkali kebutuhan data medis tersebut sangat tinggi dan membutuhkan waktu akses yang cepat sehingga pencarian data medis tidak lagi dapat dilakukan dengan algoritma pencarian *string brute-force* biasa. Ada dua algoritma *exact string matching* yang akan digunakan untuk mengatasi masalah ini, yaitu algoritma Boyer-Moore dan algoritma Horspool. Makalah ini akan menjelaskan secara ringkas tentang kedua algoritma ini dan perbandingan hasil pencarian *string* dalam bahasa medis dengan menggunakan kedua algoritma tersebut.**

***Kata Kunci*—pencarian *string,* medis, Boyer-Moore, Horspool.**

I. PENDAHULUAN

Pengetahuan dalam dunia medis telah berkembang dengan sangat pesat. Karenanya informasi dan literatur dalam dunia medis juga bertambah dengan cepat. Teknologi yang ada saat ini, seperti internet, juga sangat mendukung penyebaran informasi dan literatur medis ini. Selain literatur medis, data medis lainnya seperti data pasien juga telah mudah diakses dengan bantuan teknologi yang sekarang dikenal dengan *Electronic Patient Record* (EPR).

Kebutuhan dunia medis akan data tersebut sangatlah besar. Dunia medis juga menuntut filterisasi informasi dan kecepatan akses karena menyangkut nyawa seseorang. Karenanya data medis tersebut harus dapat diakses dengan mudah dan cepat oleh pihak yang membutuhkan. Pencarian data pun harus dapat dilakukan dengan cepat dan *real time*.

Misal ada pasien yang dalam keadaan darurat, dokter harus dapat mengakses EPR dengan cepat untuk mengetahui apakah pasien tersebut alergi terhadap obat tertentu atau tidak. Dokter juga harus dengan cepat mengetahui riwayat kesehatan seorang pasien agar dapat memberikan perawatan yang tepat.

Selain meningkatkan kinerja perangkat keras yang digunakan, kebutuhan pencarian data yang cepat ini juga dapat diatasi dengan menggunakan algoritma pencarian data yang lebih baik. Dengan menggunakan algoritma pencarian *string* yang baik, waktu pencarian data akan dapat berkurang sehingga data dapat diakses dengan lebih cepat.

Yang menarik pada kasus ini adalah bahasa medis sangatlah berbeda dengan bahasa sehari-hari. Kosakata pada bahasa medis banyak menggunakan bahasa Latin dan *Greek* yang rumit dan seringkali terdiri dari banyak karakter per katanya. Jika pencarian *string* dalam bahasa medis masih dilakukan dengan algoritma *brute-force* biasa maka akan memakan banyak waktu dan jumlah perbandingan karakter yang cenderung lebih banyak dibandingkan dengan pencarian dengan algoritma pencarian *string* lain.

Algoritma pencarian *string* yang dapat dijadikan alternatif untuk permasalahan ini adalah algoritma Boyer-Moore (BM) dan algoritma Horspool. Berbeda dengan algoritma *brute-force*, kedua algoritma ini tidak selalu membandingkan karakter per karakter pada setiap pencarian *string*. Dengan *pattern* tertentu, kedua algoritma ini dapat melakukan lompatan pengecekan karakter jika ditemukan kesalahan pada suatu kali pengecekan. Dengan metode ini, proses pencarian *string* pun dapat dilakukan dengan lebih cepat jika dibandingkan dengan pencarian *string* dengan algoritma *brute-force*.

II. ALGORITMA BOYER-MOORE

Algoritma Boyer-Moore telah dibuktikan sebagai salah satu algoritma yang paling efisien dalam aplikasi pencarian *string* dengan menggunakan *natural language* (bukan *binary language*). Algoritma ini telah sering diimplementasikan untuk fungsi "*search*" dan "*substitute*" pada *text editor*. Pada dasarnya cara kerja algoritma ini mirip dengan algoritma Knuth-Morris-Pratt (KMP) dimana kedua algoritma ini akan melakukan lompatan pengecekan sejauh mungkin dalam proses pencarian *string*. Namun berbeda dengan algoritma KMP, algoritma Boyer-Moore ini melakukan perbandingan *pattern* mulai dari kanan ke kiri.

Algoritma Boyer-Moore ini memiliki dua teknik dasar yaitu:

1. Teknik *looking-glass*
   Mencari setiap karakter dalam *pattern P* pada teks

T dengan melakukan penegcekan mundur, dimulai dari akhir P.

2. Teknik *character-jump*
   Ketika terjadi ketidakcocokan karakter pada teks dan *pattern* maka akan dilakukan lompatan "posisi" pengecekan *pattern* terhadap teks.

Algoritma Boyer-Moore memiliki fungsi praproses untuk *pattern* yang akan dicari. Fungsi ini memetakan setiap karakter pada *pattern* dengan posisi *right-most*-nya pada *pattern* tersebut. Hal ini dilakukan karena algoritma Boyer-Moore ini melakukan pengecekan *string* dari akhir *pattern*.

Misal *pattern P* = `gcagagag` , maka hasil fungsi praproses *F(x)* adalah sebagai berikut:

P	g	c	a	g	a	g	a	g
	7	6	5	4	3	2	1	0

P[*j*]	g	c	a
F(*j*)	2	6	1

Fungsi praproses yang digunakan ini sering disebut dengan metode *bad-character shift*. Selain metode ini, ada juga metode lain yaitu *good-suffix shift.* Namun metode ini tidak digunakan dan tidak akan dibahas lebih lanjut pada makalah ini.

Secara garis besar algoritma Boyer-Moore dapat dijelaskan dengan *pseudo-code* berikut:

```
function prepocessBM (input: String pattern) → array of integer
{Membuat tabel F(x) dari pattern x}

Deklarasi
Array of integer f[]

Algoritma
for i=0 step 1 to 128
   f[i] ← -1 {inisialisasi array}
endfor
for i=0 step 1 to pattern.length
   f[x.charAt(i)] ← i
endfor
→ f
```

```
function searchBM (String text, String pat) → integer
{Mengembalikan indeks ditemukannya pattern pada teks. Jika pattern tidak ditemukan maka akan mengembalikan nilai -1}

Deklarasi
Array of integer f[]
integer t = text.length
integer p = pat.length
integer i = p-1
integer pp {menyimpan nilai preprocessBM}
```

```
Algoritma
f =  preprocessBM(pattern)
if p > t then
{pattern lebih panjang dari teks}
   → -1
else
   integer j = p-1
   while i<=t-1 do
      if pat.charAt(j)=text.charAt(i) then
         if j=0 then
            → i {string cocok diposisi i}
         else {melakukan pengecekan mundur}
            i ← i-1
            j ← j-1
         endif
      else
         pp ← preprocessBM[text.charAt(i)]
         i ← i + p – min(j, 1+pp)
         j ← m – 1
      endif
   endwhile
→ -1 {tidak menemukan string yang cocok}
```

Algoritma Boyer-Moore ini memiliki kompleksitas rata-rata yaitu *O(n)* sedangkan waktu pemrosesan fungsi praprosesnya adalah *O(m+σ)*. Namun pada keadaan terbaiknya, algoritma Boyer-Moore dapat melakukan hanya *O(n/m)* perbandingan, yang merupakan jumlah minimum untuk algoritma pencarian *string* manapun yang hanya melakukan praproses pada *pattern* saja.

III. ALGORITMA HORSPOOL

A. Sejarah

Walaupun algoritma Boyer-Moore dianggap sebagai algoritma pencarian *string* yang paling efisien (untuk aplikasi umum) saat ini, namun algoritma ini ternyata kurang cocok digunakan untuk teks dengan varian karakter yang sedikit, seperti *binary string*. Hal ini dikarenakan lompatan pengecekan yang dapat dilakukan oleh algoritma Boyer-Moore menjadi cenderung lebih pendek pada teks model ini.

R. Nigel Horspool melakukan penelitian tentang algoritma Boyer-Moore ini [3]. Dari hasil penelitiannya, beliau mengemukakan gagasan tambahan untuk algoritma Boyer-Moore. Algoritma ini kemudian dikenal dengan algoritma Horspool atau algoritma Boyer-Moore-Horspool.

B. Prinsip Kerja Algoritma Horspool

Algoritma Horspool ini merupakan penyederhanaan dari algoritma Boyer-Moore. Ada dua gagasan metode algoritma yang dikemukakan oleh Horspool. Metode pertama yaitu *scan for lowest frequency character*. Maksud dari metode ini adalah dengan melakukan pencarian pada karakter yang paling jarang muncul pada teks, seperti abjad "X" atau "Q". Namun metode ini tidak akan digunakan dan tidak dibahas lebih lanjut pada makalah.

Metode kedua, yaitu metode yang digunakan pada makalah ini, adalah metode *simplified* Boyer-Moore. Jika pada algoritma Boyer-Moore terdapat dua fungsi praproses yaitu *bad-character shift* dan *good-suffix shift*, algoritma Horspool hanya menggunakan metode *bad-character shift* sebagai satu-satunya fungsi praproses *pattern* sebelum dilakukan pencarian. Hal ini dikarenakan fungsi *good-suffix shift* cenderung lebih rumit untuk diimplementasikan dan tidak terlalu berpengaruh pada kecepatan pengecekan. Fungsi *good-suffix shift* ini dibuat hanya untuk menangani *pattern* yang bersifat repetitif. Namun karena *pattern* model ini jarang ditemui maka fungsi ini tidak digunakan oleh Horspool.

Fungsi praproses *pattern* yang digunakan oleh Horspool hampir sama dengan algoritma Boyer-Moore. Namun Horspool tidak melakukan lompatan berdasarkan "*bad character*" atau karakter pada *pattern* yang ditemukan tidak cocok pada teks. Horspool menggunakan karakter paling kanan pada *current text window* untuk menentukan jarak *shift* yang akan dilakukan. *Pattern* akan digeser sampai karakter paling kanan dari ujung *window* pengecekan yang benar. Karakter yang berada pada ujung *pattern* tidak dihitung dan tidak dijadikan karakter terkanan dari karakter yang sama dengannya.

Contoh:

0	1	2	3	4	5	6	7	8	9	...
a	b	d	v	e	d	v	v	e	g	a
e	d	v	v	e						
	e	d	v	v	e	dst				

Algoritma Boyer-Moore

0	1	2	3	4	5	6	7	8	9	...
a	b	d	v	e	d	v	v	e	g	a
e	d	v	v	e						
				e	d	v	v	e	dst	

Algoritma Horspool

Secara garis besar algoritma Horspool dapat dijelaskan dengan *pseudo-code* berikut:

```
function Horspool (String text, String pat)
→ integer
{Mengembalikan indeks ditemukannya pattern
pada teks. Jika pattern tidak ditemukan
maka akan mengembalikan nilai -1}

Deklarasi
Array of integer f[]
integer t = text.length
integer p = pat.length
integer i = 0
char c


Algoritma
f =  preprocessBM(pattern)
while i<=t-p do
   c = text.charAt(i+p-1)
   if pat.charAt(p-1)=c and
   pencarian mencapai akhir pattern then
      → i {string cocok diposisi i}
   else
      i ← i+f(text.charAt(c))
endwhile
→ -1 {tidak menemukan string yang cocok}
```

Algoritma Horspool ini memiliki kompleksitas rata-rata sama dengan algoritma Boyer-Moore yaitu *O(n)* sedangkan waktu pemrosesan fungsi praprosesnya adalah *O(m+σ)*.

IV. KARAKTERISTIK BAHASA MEDIS

Pada umumnya, istilah-istilah pada bahasa medis menggunakan kosakata dari bahasa Latin dan *Greek* yang cukup panjang dan rumit. Istilah pada bahasa medis biasanya terbagi menjadi tiga bagian besar, yaitu root, akhiran (*suffix*), dan awalan (*prefix*). Ketiga bagian ini banyak menggunakan bahasa Latin atau *Greek* sehingga banyak kata pada bahasa medis yang memiliki awalan, root, atau akhiran yang sama.

Hal ini kurang menguntungkan untuk pencarian *string* yang dilakukan per karakter secara sekuensial karena kemungkinan terjadinya kegagalan cek sangat besar. Sebagai contoh kata `polyneuropathy` yang terdiri dari tiga bagian utama yaitu `poly`, `neuro`, dan `pathy`. Berdasarkan kamus istilah medis [7], berikut adalah tabel kemunculan kata `polyneuropathy`, *suffix* dan *prefix*nya beserta minimum iterasi yang diperlukan sebelum pengecekan kecocokan *string* berhasil:

Tabel 1 Jumlah kemunculan kata di kamus dan jumlah iterasi pengecekan minimum

Kata	Kemunculan pada kamus	Iterasi minimum yang diperlukan
`Polyneuropathy`	3	42
`Poly`	173	692
`Neuropathy`	29	290
`Neuro`	379	1895
`Pathy`	126	630

Jika pencarian *string* `polyneuropathy` dilakukan per satu karakter dari kiri ke kanan seperti menggunakan algoritma *brute-force*, maka algoritma tersebut harus memeriksa 173 kata `poly` dengan total 692 iterasi sebelum akhirnya mengecek kata `neuro` dan `pathy`. Karena itu, algoritma lain perlu diterapkan untuk pemecahan masalah ini.

V. HASIL *STRING MATCHING* ALGORITMA BM DAN HORSPOOL PADA BAHASA MEDIS

Percobaan pencarian *string* dilakukan dengan menggunakan bahasa pemrograman Java. File teks yang akan dijadikan bahan pencarian adalah buku *Black's Medical Dictionary* [7] yang terdiri dari 3471943 karakter.

Ada tiga *pattern* yang dipilih sebagai bahan percobaan, yaitu:

1. `Fossa`
   Pattern ini dipilih karena hanya terdiri dari sedikit karakter (5 karakter) dan tidak memiliki awalan dan akhiran yang umum pada bahasa medis.
2. `Polyneuropathy`
   Pattern ini dipilih karena terdiri dari cukup banyak karakter (14 karakter). Selain itu, awalan `poly-`, akhiran `pathy-`, dan *root* `neuro` cukup banyak digunakan pada bahasa medis.
3. `Other causes of left ventricular failure`
   Pattern ini dipilih karena terdiri dari banyak karakter (40 karakter) dan terdiri dari kata-kata yang umum ditemukan pada kamus medis.

Hasil percobaan pencarian *string* dengan ketiga buah pattern diatas adalah sebagai berikut (waktu percobaan diabaikan):

1. *Pattern:* `Fossa`

```
run:
Boyer-Moore
Pattern: fossa
Text length: 3471943
Comparison: 798103
Found at: [10598, 41284, 41813, 221628, 1221369, 1221421,
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 1 Hasil pencarian *pattern* ke-1 dengan algoritma Boyer-Moore

```
run:
Horspool
Pattern: fossa
Text length: 3471943
Comparison: 1488534
Found at: [10598 41284 41813 221628 1221369 1221421
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 2 Hasil pencarian pattern ke-1 dengan algoritma Horspool

2. *Pattern:* `Polyneuropathy`

```
run:
Boyer-Moore
Pattern: polyneuropathy
Text length: 3471943
Comparison: 352760
Found at: [853022, 2140919, 2141219]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 3 Hasil pencarian pattern ke-2 dengan algoritma Boyer-Moore

```
run:
Horspool
Pattern: polyneuropathy
Text length: 3471943
Comparison: 693662
Found at: [853022 2140919 2141219 ]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 4 Hasil pencarian pattern ke-2 dengan algoritma Horspool

3. *Pattern:* `Other causes of left ventricular failure`

```
run:
Boyer-Moore
Pattern: other causes of left ventricular failure
Text length: 3471943
Comparison: 194514
Found at: [2213106]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Gambar 5 Hasil pencarian pattern ke-3 dengan algoritma Boyer-Moore

```
run:
Horspool
Pattern: other causes of left ventricular failure
Text length: 3471943
Comparison: 355302
Found at: [2213106 ]
BUILD SUCCESSFUL (total time: 1 second)
```

Gambar 6 Hasil pencarian pattern ke-3 dengan algoritma Horspool

VI. ANALISIS HASIL PERCOBAAN

Untuk mempermudah analisis, hasil percobaan yang telah dilampirkan di bab sebelumnya disusun menjadi sebuah tabel dibawah ini.

Tabel 2 Jumlah pengecekan karakter yang dilakukan untuk setiap *pattern*

Pattern	Jml Karakter	Boyer-Moore	Horspool
`Fossa`	5	798103	1488534
`Polyneuropathy`	14	352760	693662
`Other causes of left ventricular failure`	40	194514	355302

Dari tabel dapat dilihat bahwa pada ketiga pengujian jumlah pengecekan karakter yang dilakukan oleh algoritma Boyer-Moore lebih sedikit dibandingkan dengan jumlah pengecekan yang dilakukan oleh algoritma Horspool.

Seperti telah dibahas pada bab IV, bahasa medis banyak menggunakan kosakata yang panjang dengan varian karakter yang beragam. Karakter teks seperti bahasa medis ini sangat cocok dengan algoritma Boyer-Moore. Dengan banyaknya varian karakter pada teks dan *pattern*, maka ketidakcocokan teks dan *pattern* akan lebih

mudah terdeteksi pada awal pengecekan sehingga algoritma dapat langsung melakukan lompatan. Hal ini akan mengurangi jumlah pengecekan karakter yang harus dilakukan oleh karakter.

Dari tabel juga dapat diamati bahwa semakin panjang *pattern* yang dicari maka semakin sedikit jumlah pengecekan karakter yang dilakukan oleh kedua algoritma ini. Hal ini dikarenakan semakin panjang *pattern* yang dicari maka semakin besar kemungkinan kesamaan karakter dalam *pattern*. Karena itu lompatan pengecekan yang dapat dilakukan oleh kedua algoritma akan semakin besar.

Waktu percobaan memang tidak diukur secara akurat pada percobaan ini. Namun dari hasil yang terlampir saat melakukan kompilasi program, waktu percobaan rata-rata kurang dari satu detik untuk kedua algoritma dengan panjang teks yaitu 3471943 karakter. Berarti kedua algoritma tersebut kira-kira dapat melakukan pencarian sebanyak 3,4 juta karakter/detik.

VII. KESIMPULAN

Algoritma Boyer-Moore adalah salah satu algoritma pencarian *string* yang paling efisien yang digunakan untuk berbagai aplikasi umum saat ini. Kelebihan algoritma ini dibandingkan algoritma pencarian *string brute-force* adalah algoritma ini dapat melakukan lompatan pengecekan jika terjadi ketidakcocokan karakter antara *pattern* dan teks. Dengan metode ini, jumlah pengecekan karakter yang diperlukan akan menjadi lebih sedikit dan waktu pencarian akan menjadi lebih cepat.

Algoritma Horspool adalah penyederhanaan algoritma Boyer-Moore yang dibuat oleh R. Nigel Horspool. Algoritma ini dibuat karena algoritma Boyer-Moore yang kurang cocok diimplementasikan untuk pencarian *string* dengan varian karakter yang sedikit seperti *binary string*. Algoritma ini bekerja dengan metode yang hampir sama dengan algoritma Boyer-Moore namun tidak melakukan lompatan berdasarkan karakter pada *pattern* yang ditemukan tidak cocok pada teks. Horspool menggunakan karakter paling kanan pada *current text window* untuk menentukan jarak *shift* yang akan dilakukan

Dari percobaan, didapatkan hasil bahwa pencarian *string* pada teks bahasa medis dengan menggunakan algoritma Boyer-Moore lebih baik dibandingkan dengan algoritma Horspool. Bahasa medis banyak menggunakan kosakata yang panjang dengan varian karakter yang beragam. Karakter teks seperti bahasa medis ini sangat cocok dengan algoritma Boyer-Moore sehingga algoritma ini dapat bekerja dengan maksimal pada pencarian *string* dengan teks menggunakan bahasa medis seperti ini.

REFERENSI

[1] Munir, Rinaldi, *Diktat Kuliah IF3051 Strategi Algoritima*. Bandung: Program Studi Teknik Informatika STEI ITB, 2009, pp. 186-193.

[2] Lovis C, Baud R, *Fast Exact String Pattern-matching Algorithm Adapted to the Characteristics of The Medical Language*. American Medical Informatics Association, 2000. http://www.ncbi.nlm.nih.gov/pmc/articles/PMC61442/, waktu akses: 05 Desember 2011 pukul 22.00.

[3] Horspool, R. Nigel, *Practical Fast Searching in Strings*. Software-Practice and Experience Vol.10, 501-506 (1980).

[4] Charras C, Lecroq T, *Exact String Matching Algorithm*. http://www-igm.univ-mlv.fr/~lecroq/string/, waktu akses: 05 Desember 2011 pukul 21.00.

[5] *Horspool Algorithm.* http://www.inf.fhflensburg.de/lang/algorithmen/pattern/horsen.htm, waktu akses: 06 Desember 2011 pukul 18.00.

[6] Davison, Andrew, *Pattern Matching*.

[7] Marcovitch, Harvey, *Black's Medical Dictionary 41st Edition*. London: A&C Black, 2005.

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, 29 April 2010

Evlyn Dwi Tambun / 13509084