Penggunaan Algoritma Boyer-Moore pada Instruksi-instruksi GNU Grep

Muhammad Gumilang / 13514092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
muhammadgumilang11@gmail.com

***Abstrak*—Semakin canggihnya zaman terindikasi dengan peningkatan performansi pada perangkat lunak. Salah satu cara mingingkatkan performansi software adalah dengan menggunakan algoritma yang sangkil dan mangkus. Salah satu hal pada dunia teknologi yang menggunakan algoritma sangkil dan mangkus adalah GNU. Salah satu perangkat lunak turunan GNU adalah grep. Grep digunakan untuk mencari suatu pattern berupa potongan teks yang diinginkan untuk dicari pada suatu file teks. Grep merupakan string pattern matching yang menggunakan algoritma Boyer-Moore. Grep memiliki instruksi-instruksi untuk mencari *pattern* yang diinginkan. Pada makalah ini akan ditunjukkan penggunaan algoritma Boyer-Moore pada instruksi-instruksi grep yang membuat grep cepat dalam mengeksekusi instruksi.**

***Keywords*—GNU, Grep, Boyer-Moore, *regular expressions*, teks, *pattern*.**

I. PENDHALUAN

GNU merupakan suatu sistem operasi komputer yang sepenuhnya terdiri dari perangkat-perangkat lunak bebas. Namanya merupakan akronim berulang untuk *GNU's Not UNIX* (GNU bukanlah UNIX) yang dipilih karena rancangannya mirip UNIX, tetapi berbeda dengan UNIX, GNU tidak mengandung kode-kode UNIX. GNU dikembangkan oleh Richard Stallman.

Varian dari sistem operasi GNU, yang menggunakan kernel Linux, dewasa ini telah digunakan secara meluas. Walau terkadang dirujuk sebagai 'Linux', sebetulnya lebih tepat jika disebut sistem *GNU/Linux*. Salah satu fitur yang sangat menarik dari GNU/Linux yang belum ada di sistem operasi populer lainnya, yaitu menjalankan sistem operasi dan aplikasi lengkap tanpa menginstalnya di hard disk. Hal ini memudahkan kita menggunakan GNU/Linux di komputer milik orang lain karena tak perlu menginstal atau mengutak-atik hard disk dan partisinya.

Proyek GNU diluncurkan pertama kali pada tahun 1984. Proyek ini diawali pada tahun 1983 dan diketuai oleh Richard Stallman untuk membuat sistem operasi seperti UNIX lengkap, yaitu kompiler, utiliti aplikasi, utiliti pembuatan, dan lain-lain yang sepenuhnya diciptakan dengan perangkat lunak bebas. Pada tahun 1991, pada tahap versi pertama kerangka Linux ditulis, proyek GNU telah menghasilkaan seluruh komponen sistem, kecuali kernal. Torvalds dan pembuat kernel seperi Linux menyesuaikan kernelnya suapaya dapat bekerja pada komponen GNU. Oleh karena itu, Linux melengkapi ruang terakhir dalam rancangan GNU.

Grep merupakan utiliti *command-line* untuk mencari kumpulan data *plain-text* yang sesuai dengan sebuah *regular expression*. Grep pertama kali dikembangkan untuk sistem operasi UNIX, namun sekarang tersedia untuk seluruh sistem UNIX-like atau sistem-sistem seperti UNIX. Nama Grep berasal dari perintah *ed g/re/p*, ***g****lobally search a* ***r****egular* ***e****xpression and* ***p****rint*, yang berfungsi untuk tentunya melakukan pencarian secara global dengan regular ekspresi yang ada dan mencetak semua baris yang bersesuaian.

GNU grep tergolong cepat karena perangkat lunak ini menghindara pemeriksaan pada setiap byte yang dimasukkan. Selain itu, GNU grep mengekskusi sangat sedikit instruksi untuk setiap byte yang diperiksa. GNU grep menggunakan algoritma yang terkenal Boyer-Moore, dimana pemeriksaan pertama pada huruf terakhir pada pattern, dan menggunakan tabel untuk memberitahu sejauh apa teks dapat diloncati apabila ditemukan karakter yang tidak bersesuaian.

GNU grep juga membeberkan bagian dalam loop Boyer-Moore, dan menyiapkan tabel masukan Boyer-Moore sedemikian rupa sehingga tidak diperlukan *loop exit test* untuk setiap tahap yang dibeberkan. Hasil dari itu, dalam batas, GNU grep memiliki rata-rata instruksi lebih kecil dari 3 x86 instruksi tereksekusi untuk setiap byte yang diperiksa.

Jadi, dari pada menggunakan *line-oriented* input, GNU grep membaca data mentah ke dalam *buffer* yang besar. Dan pada *buffer* tersebut digunakan algoritma Boyer-Moore untuk mencari pattern yang diinginkan pada text file yang diberikan.

Grep memiliki instruksi-instruksi untuk memanipulasi pengambilan hasil dari pencarian *pattern*. Instruksi-instruksi tersebut digambarkan dalam *regular expressions* dan *extended regular expressions*, dimana dari tiap-tiap instruksi tersebut tentunya menggunakan algoritma yang efisien untuk menjalankannya.

II. LANDASAN TEORI

2.1 String Pattern Matching

String pattern matching merupakan prses pencocokkan *string* pada sebuah teks yang diberikan. Persoalan pencarian *string* dapat dijabarkan sebagai berikut.

Diberikan:

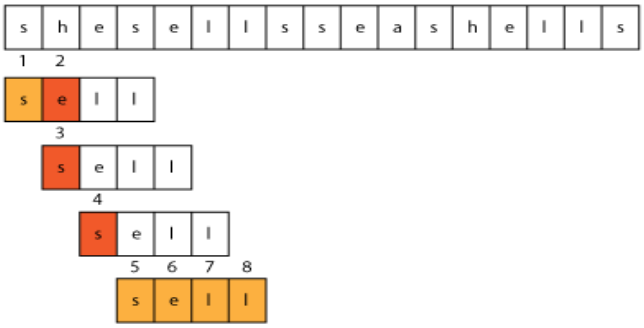
1. Teks (*text*), yaitu long string yang panjangnya *n* karakter.
2. *Pattern*, yaitu string dengan panjang *m* karakter (*m* < *n*) yang akan dicari di dalam teks.

Dicari di dalam teks yang bersesuaian dengan *pattern.*

Berikut merupakan contoh teks dan *pattern* yang ingin dicari dan pencariannya dengan algoritma naive/Brute Froce.

Teks (T) : "shesellsseashells"

Pattern (P) : "sell"



Gambar 1. *String pattern matching* dengan algoritma Brute Force

2.2 Algoritma Boyer-Moore

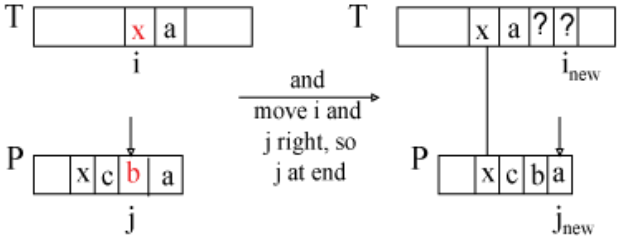
Algoritma Boyer-Moore merupakan salah satu algoritma pencarian *string* yang dipublikasikan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Aplikasi ini dianggap sebagai algoritma paling efisien pada aplikasi umum. Yang membedakan algoritma Boyer-Moore dari algoritma lain adalah cara memeriksa *pattern* algoritma Boyer-Moore yang dimulai dari kanan *pattern*. Hal ini ditujukan untuk mendapat lebih banyak informasi dengan memulai pencocokan karakter dari kana, dan bukan dari kiri. Namun ide utama pada algoritma ini adalah cara pencarian *string* dengan malompat sjauh mungkin untuk mengurangi jumlah pergeseran.

Dengan menggunakan algoritma ini, secara rata-rata proses pencarian akan menjadi lebih cepat jika dibandingkan dengan menggunakan algoritma lainnya. Hal ini dikarenakan algoritma ini mencocokkan string dari pattern yang paling belakang, jika tidak sama maka akan langsung melompat mencari karakter yang cocok.

Pada algoritma Boyer-Moore digunakan dua teknik, yaitu teknik *looking-glass* dan teknik *character-jump*. Teknik *looking glass* merupakan teknik pencarian *pattern* pada teks dengan pencocokan dimulai dari akhir *pattern* pada teks dengan pencocokan dimulai dari akhir *string* pada *pattern* ke depan. Sedangkan teknik *character-jump* merupakan teknik menggeser pattern ketika tidak terjadi kecocokan (*mismatch*) sejauh karakter tertentu bergantung pada bagaimana ketidakcocokan itu terjadi. Karena hal itu, terdapat tiga kasus khusus dalam algoritma Boyer-Moore yang menandakan bagaimana ketidakcocokan itu terjadi.

a) Kasus pertama

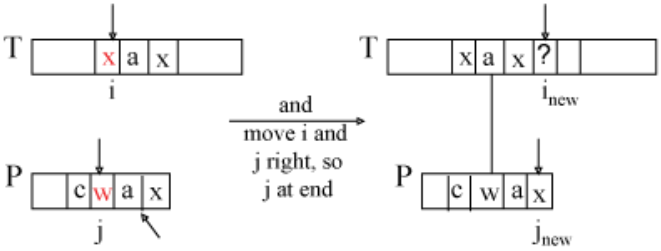
Jika suatu *pattern* P mengandung *x* disuatu tempat, maka coba menggeser P ke kanan sejauh mungkin hingga kemunculan *x* terakhir di P bersesuaian dengan T[i].



Gambar 2. Kasus pertama *mismatch*

b) Kasus kedua

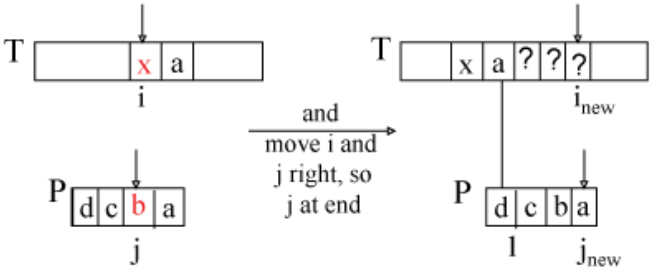
Jika P mengandung *x* di suatu tempat, tetapi dengan penggeseran P ke kanan menuju kemunculan terakhir *x* tidak memungkinkan, maka geser P ke kanan sejauh 1 karakter ke T[i+1].



Gambar 3. Kasus kedua *mismatch*

c) Kasus ketiga

Jika kasus kesatu dan kasus kedua tidak memungkinkan, maka geser P ke kanan sehingga P[i] bersesuaian dengan T[i+1].



Gambar 4. Kasus ketiga *mismatch*

Sebelum melakukan pencocokan teks dengan algoritma Boyer-Moore, terdapat pra-proses yang harus diimplementasikan agar penggunaan algoritma ini semakin mudah. Proses ini dimakan sebagai fungsi kemunculan terakhir (*last occurence*), dimana fungsi ini beruguna untuk menentukan kemunculan terakhir dari setiap karakter pada pattern dan kamudian disimpan pada sebuah array. Dengan fungsi kemunculan terakhir ini, kita dapat menentukan pergeseran *pattern* dengan lebih mudah tanpa harus menghitung kembali untuk setiap ketidakcocokan karakter

yang terjadi antara teks dan *pattern*. Berikut contoh tabel fungsi kemunculan terakhir dari *pattern* "SHELL"

x	s	h	e	l
L(x)	1	2	3	5

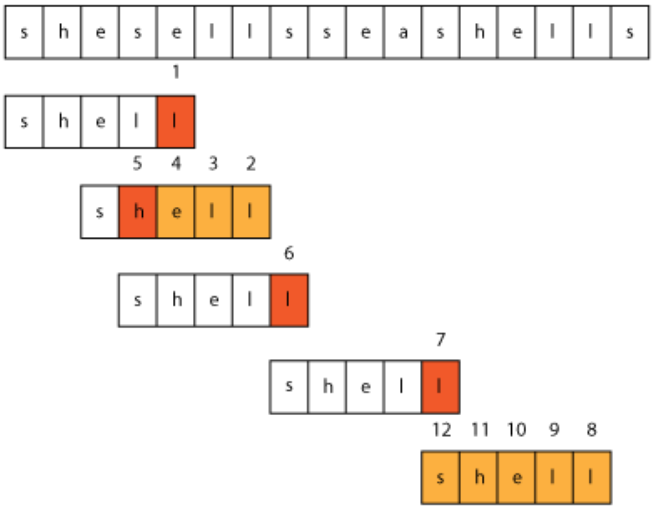
Gambar 5. Tabel fungsi kemunculan terakhir

Berikut adalah contoh pencarian teks dan *pattern* dengan algoritma Boyer-Moore.

Teks (T) : "SHESELLSSEASHELLS"

Pattern (P) : "SHELL"

Maka pencarian menggunakan algoritma Boyer-Moore adalah sebagai berikut.



Gambar 6. *String pattern matching* dengan algoritma Boyer-Moore

Berdasarkan gambar di atas dapat dilihat bahwa dengan menggunakan algoritma Boyer-Moore dengan teks dan *pattern* yang telah disebutkan sebelumnya, maka terdapat total 12 jumlah perbandingan. Jumlah perbandingan dengan algoritma Boyer-Moore bila dibandingkan dengan algoritma lainnya seperti Brute Force atau Knuth-Morris-Pratt jauh lebih sedikit karena algoritma Boyer-Moore tidak membandingkan *string* dari kiri ke kanan, melainkan sebaliknya.

Pada pemeriksaan pertama, terjadi ketidakcocokan karakter. Karena ketidakcocokan pada huruf 'e' dimana huruf tersebut ada pada *pattern*, maka geser *pattern* sejauh 3 karakter sesuai dengan tabel fungsi kemunculan terakhir pada gambar 5. Hal tersebut termauk ke dalam kasus kesatu. Pencarian dilanjutkan hingga selesai pada langkah ke-12.

```
public static int bmMatch(String text,
                          String pattern)
{
  int last[] = buildLast(pattern);
  int n = text.length();
  int m = pattern.length();
  int i = m-1;

  if (i > n-1)
    return -1; // no match if pattern is
               // longer than text
  int j = m-1;
  do {
    if (pattern.charAt(j) == text.charAt(i))
      if (j == 0)
        return i; // match
      else { // looking-glass technique
        i--;
        j--;
      }
    else { // character jump technique
       int lo = last[text.charAt(i)];  //last occ
       i = i + m - Math.min(j, 1+lo);
       j = m - 1;
    }
  } while (i <= n-1);

  return -1; // no match
} // end of bmMatch()

public static int[] buildLast(String pattern)
/* Return array storing index of last
  occurrence of each ASCII char in pattern. */
{
  int last[] = new int[128]; // ASCII char set

  for(int i=0; i < 128; i++)
    last[i] = -1; // initialize array

  for (int i = 0; i < pattern.length(); i++)
    last[pattern.charAt(i)] = i;

  return last;
} // end of buildLast()
```

Gambar 7. Implementasi algoritma Boyer-Moore dalam bahasa Java

Kompleksitas waktu pencarian *string* dengan menggunakan algoritma Boyer-Moore dalam kasus terburuknya (*worst case*) adalah O(*mn*+*A*) dengan *A* merupakan jumlah alphabet yang digunakan.

III. Analisis

GNU grep menyediakan beberapa instruksi untuk mencari *pattern* yang diinginkan pada teks yang ada. Tentunya dengan instruksi-instruksi berbeda tersebut, dengan algoritma Boyer-Moore membuat GNU grep bekerja dengan cepat.

3.1 Penggunaan Dasar

Pada bentuk tersimpelnya, grep dapat digunakan untuk mencocokan *literal patterns* pada sebuah file teks. Hal ini berarti juka kita memberikan grep sebuah kata untuk dicari, grep akan mencetak setiap baris yang mengandung kata tersebut.

Berikut adalah contoh. Kita akan menggunakan grep untuk mencari setiap baris yang mengandung kata “GNU” pada *GNU General Public License* versi 3 pada sistem Ubuntu.

```
grep "GNU" GPL-3
                    GNU GENERAL PUBLIC LICENSE
  The GNU General Public License is a free, copyleft license for
the GNU General Public License is intended to guarantee your freedom to
GNU General Public License for most of our software; it applies also to
  Developers that use the GNU GPL protect your rights with two steps:
  "This License" refers to version 3 of the GNU General Public License.
  13. Use with the GNU Affero General Public License.
under version 3 of the GNU Affero General Public License into a single
...
...
```

Argumen pertama, “GNU”, adalah *pattern* yang sedang kita cari, sedangkan argumen kedua, ”GPL-3”, adalah file input yang ingin kita cari. Hasil output akan berupa setiap baris yang mengandung *pattern* pada teks. Dalam beberapa distribusi Linux, *pattern* yang dicari akan di-*highlight* pada baris hasil.

Berikut adalah penggambaran pencarian *pattern* “GNU” pada teks “GPL-3” dengan algoritma Boyer-Moore.

```
13. Use with the GNU Affero General Public License.
GNU   GNU   GNU  GNU
   GNU   GNU   GNU
```

Gambar 8. Pencarian *pattern* “GNU” dengan algoritma Boyer-Moore

3.2 Opsi Umum

Secara *default*, grep hanya akan mencari untuk *pattern* yang eksak sesuai dengan yang diberikan pada file input dan mengembalikan baris yang ditemukan. Kita dapat sifat ini menjadi lebih berguna dengan menambah instruksi-instruksi sebagai opsi umum.

Apabila kita ingin grep untuk melakukan pencarian dengan tidak memperdulikan variasi *upper-* dan *lower-case* kata pada teks, kita dapat menambahkan opsi “-i” atau “--ignore-case”.

```
grep -i "license" GPL-3
                    GNU GENERAL PUBLIC LICENSE
 of this license document, but changing it is not allowed.
  The GNU General Public License is a free, copyleft license for
  The licenses for most software and other practical works are designed
the GNU General Public License is intended to guarantee your freedom to
GNU General Public License for most of our software; it applies also to
price.  Our General Public Licenses are designed to make sure that you
(1) assert copyright on the software, and (2) offer you this License
  "This License" refers to version 3 of the GNU General Public License.
  "The Program" refers to any copyrightable work licensed under this
...
...
```

Dapat dilihat, kita mendapatkan hasil yang mengandung: “LICENSE”, “lincense”, dan “License”. Apabila ada kata “LiCeNsE”, maka kata tersebut juga akan dicetak beserta barisnya.

Berikut pencarian perintah di atas dengan algoritma Boyer-Moore dengan menon-kapitalkan baris yang dicari terlebih dahulu.

```
"this license" refers to version 3 of the gnu general public license.
license
      license
```

Gambar 9. Pencarian *pattern* ”license”, *ignore-case*, dengan algoritma Boyer-Moore

Selain *ignore case*, GNU grep juga menyediakan opsi apabil kita ingin mencari baris yang tidak mengandung *pattern* yang diberikan, yaitu “-v” atau “--invert-match”.

```
grep -v "the" BSD
All rights reserved.

Redistribution and use in source and binary forms, with or without
are met:
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
...
...
```

Dapat dilihat, karena kita menspesifikasi opsi “ignore case”, maka pada kedua hasil terakhir dikembalikan baris sebagaimana tidak mengandung kata ”the”.

Berikut pencarian perintah di atas dengan algoritma Boyer-Moore.

```
Redistribution and use in source and binary forms, with or without
the  the  the  the  the  the  the  the  the  the  the  the
   the  the  the  the  the  the  the  the  the  the  the the
```

Gambar 10. Pencarian *pattern* “the” sehingga pada baris tersebut tidak ditemukan.

3.3 Regular Expressions

Pada pendahuluan, kita mengetahui bahwa grep merupakan singkatan dari “*global regular expression print*”. Sebuah *regular expression* adalah sebuah teks *string* yang menggambarkan *pattern* tertentu. Aplikasi dan bahas pemrograman yang berbeda mengimplentasi *regular expression* yang sedikit berbeda. Kita hanya akan membahas sebagian kecil cara grep menggambarkan *pattern*.

3.3.1 Literal Matches

Pada contoh di atas sebelumnya, saat kita mencari kata “GNU” dan “the”, kita mencari *regular expression* yang sangat sederhana, yang mencocokan *string* “GNU” dan “the” secara eksak. *Pattern* yang secara eksak menspesifikasi karakter untuk dicocokan disebut dengan “*literals*” karena *pattern* dicocokan langsung, karakter ke karakter.

3.3.2 Anchor Matches

Anchors adalah karakter spesial yang menspesifikasi dimana pada baris harus terjadi kecocokan tertentu untuk dianggap sah.

Contohnya, dengan *anchors*, kita dapat menspesifikasi bahwa kita hanya ingin mengetahui baris yang memiliki kata “GNU” pada awal baris. Untuk melakukan ini, kita hanya perlu menambahkan *anchor* “^” sebelum *literal string*.

```
grep "^GNU" GPL-3
GNU General Public License for most of our software; it applies also to
GNU General Public License, you may choose any version ever published
```

Selain itu, *anchor* “$” digunakan setelah sebuah *string* untuk mengindikasi bahwa kecocokan hanya terjadi bila pada akhir baris muncul kata yang diingin kan.

```
grep "and$" GPL-3
that there is no warranty for this free software.  For both users' and
  The precise terms and conditions for copying, distribution and
  License.  Each licensee is addressed as "you".  "Licensees" and
receive it, in any medium, provided that you conspicuously and
    alternative is allowed only occasionally and noncommercially, and
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
provisionally, unless and until the copyright holder explicitly and
receives a license from the original licensors, to run, modify and
make, use, sell, offer for sale, import and otherwise run, modify and
```

Dari kedua contoh tersebut, penggunaan algoritma manapun, Boyer-Moore atau Brute Force, tidak menjadi masalah, karena pemeriksaan hanya terjadi sekali tanpa pergeseran.

3.3.3 Matching Any Character

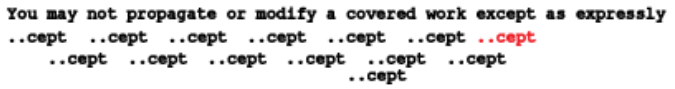
Karakter titik (.) digunakan pada *regular expression* untuk menandakan bahwa karakter apapun dapat berada pada lokasi tersebut.

Sebagai contoh, jika kita ingin mencocokan sesuatu dengan dua karakter dan dilanjutkan dengan *string* "cept", kita dapat menggunakan *pattern* sebagai berikut.

```
grep "..cept" GPL-3
use, which is precisely where it is most unacceptable.  Therefore, we
infringement under applicable copyright law, except executing it on a
tells the user that there is no warranty for the work (except to the
License by making exceptions from one or more of its conditions.
form of a separately written license, or stated as exceptions;
  You may not propagate or modify a covered work except as expressly
  9. Acceptance Not Required for Having Copies.
...
...
```

Dapat dilihat, kita mendapatkan hasil "accept" maupun "except" dan variasi dari kedua kata tersebut.

Berikut pencarian perintah di atas digambarkan dengan algoritma Boyer-Moore.



Gambar 11. Pencarian *pattern* "..cept" dengan algoritma Boyer-Moore

Pada contoh di atas, saat terjadi *mismatch* dimana huruf tersebut tidak terdapat pada "cept", *pattern* digeser hingga "." sejajar dengan huruf yang tidak cocok. Hal ini terjadi karena karakter "." dapat menjadi karakter apasaja dalam kasus ini.

3.3.4 Bracket Expressions

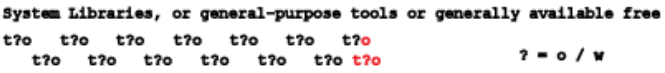
Dengan menempatkan sebuah grup karakter dalam *brackets* ("[" dan "]"), kita dapat menspesifikasi bahwa karakter pada posisi tersebut dapat berupa salah satu karakter yang terdapat pada grup *bracket*.

Ini menandakan bahwa jika kita ingin mencari baris yang mengandung kata "too" atau "two", kita dapat menspesifikasi variasi tersebut sedemikian rupa menjadi

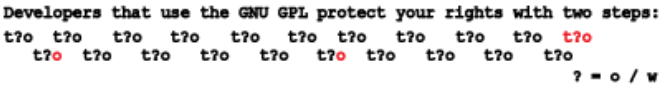
```
grep "t[wo]o" GPL-3
your programs, too.
freedoms that you received.  You must make sure that they, too, receive
  Developers that use the GNU GPL protect your rights with two steps:
a computer network, with no transfer of a copy, is not conveying.
System Libraries, or general-purpose tools or generally available free
    Corresponding Source from a network server at no charge.
...
...
```

Dapat dilihat, kedua variasi muncul dalam file. Notasi *brackte* juga memberlakukan opsi yang menarik. Kita dapat mencocokan apapun kecuali karakter yang berada pada *bracket* dengan mengawali karakter tersebut dengan karakter "^". Berikut adalah contoh *pattern* ".ode", namun tidak mencari *pattern* "code".

Berikut pencarian perintah di atas digambarkan dengan algoritma Boyer-Moore.



Gambar 12. Pencarian *pattern* "t[wo]o" dengan hasil "too" yang didapat menggunakan algoritma Boyer-Moore



Gambar 13. Pencarian *pattern* "t[wo]o" dengan hasil "two" yang didapat menggunakan algoritma Boyer-Moore

3.4 Extended Regular Expression

Grep dapat digunakan dengan *regular expression* yang lebih dalam dengan *flag* "-E" atau dengan memanggil perintah "egrep". Opsi ini membuka kemampuan dari *extended regular expressions*. *Extended regular expressions* melingkupi meta-karakter dasar, bersama dengan meta-karakter untuk mengekspresikan pencocokan yang lebih kompleks.

3.4.1 Alternation

Sama halnya dengan *bracket expression* yang dapat menspesifikasikan kemungkinan untuk pilihan satu karakter, *alternation* memperbolehkan kita untuk menspesifikasikan pencocokan alternatif untuk *string* atau himpunan ekspresi.

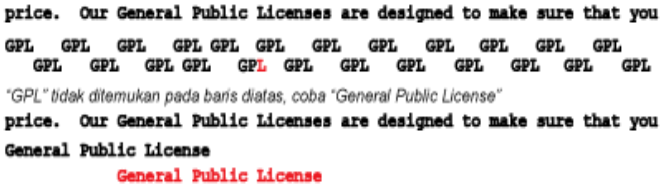
Untuk menandakan *alternation*, digunakan karakter "|". Karakter ini digunakan dalam group parenthesis untuk menandakan satu dari dua atau lebih kemungkinan dapat dianggap cocok.

Berikut contoh mencari kata "GPL" atau "General Public License" pada teks.

```
grep -E "(GPL|General Public License)" GPL-3
  The GNU General Public License is a free, copyleft license for
the GNU General Public License is intended to guarantee your freedom to
GNU General Public License for most of our software; it applies also to
price.  Our General Public Licenses are designed to make sure that you
  Developers that use the GNU GPL protect your rights with two steps:
  For the developers' and authors' protection, the GPL clearly explains
authors' sake, the GPL requires that modified versions be marked as
have designed this version of the GPL to prohibit the practice for
those
...
...
```

Alternation dapat memilih lebih dari dua pilihan dengan menambahkan karakter "|".

Berikut pencarian perintah di atas digambarkan dengan algoritma Boyer-Moore.



Gambar 14. Pencarian *pattern* "GPL" dan "General Public License" dengan algoritma Boyer-Moore

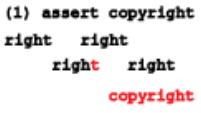
3.4.2 Quantifier

Seperti halnya dengan meta-karakter "*" yang mencocokan karakter atau himpunan karakter sebelumnya yang muncul sebanyak nol atau lebih, karakter "?" digunakan untuk menandakan karakter sebelumnya dapat muncul 0 atau 1 kali. Secara esensial, karakter ini membuat pilihan apakan karakter atau himpunan karakter ada atau tidak.

Berikut adalah contoh pencocokan kata "copyright" dan "right" dengan meletakkan "copy" sebagai grup pilihan

```
grep -E "(copy)?right" GPL-3
 Copyright (C) 2007 Free Software Foundation, Inc.
  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
know their rights.
  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
  "Copyright" also means copyright-like laws that apply to other kinds of
...
...
```

Berikut pencarian perintah di atas digambarkan dengan algoritma Boyer-Moore.



Gambar 15. Pencarian *pattern* "right" atau "copyright" dengan algoritma Boyer-Moore

Pada contoh di atas, pertama-tama dicari *pattern* "right" yang sesuai pada teks. Setelah ditemukan kecocokan *pattern* "right", periksa apakah *pattern* "copy" mengikuti sebelum *pattern* "right".

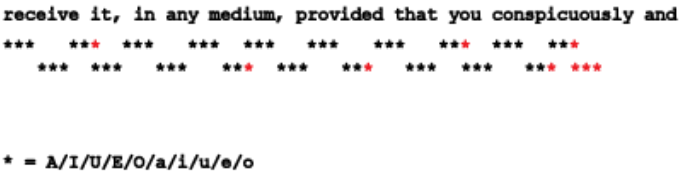
3.4.3 Specifying Match Repetition

Jika kita ingin menspesifikasikan jumlah pencocokan terulang, kita dapat menggunakan karakter *brace* ("{" dan "}"). Karakter ini digunakan untuk menentukan jumlah eksak ekspresi dapat cocok.

Jika kita ingin mencari baris yang mengandung *tripel-vowels*, huruf vokal tiga kali berturut-turut, kita dapat menggunakan ekspresi sebagai berikut.

```
grep -E "[AEIOUaeiou]{3}" GPL-3
changed, so that their problems will not be attributed erroneously to
authors of previous versions.
receive it, in any medium, provided that you conspicuously and
give under the previous paragraph, plus a right to possession of the
covered work so as to satisfy simultaneously your obligations under this
```

Berikut pencarian perintah di atas digambarkan dengan algoritma Boyer-Moore.



Gambar 16. Pencarian *pattern* 3 huruf vokal dengan algoritma Boyer-Moore

IV. KESIMPULAN

Sering terjadi dimana grep akan berguna dalam mencari *pattern* dalam suatu file. Alangkah baiknya bila kita mengenal dengan opsi dan sintaks untuk menghemat waktu. Selain mengetahui sintaks yang berguna pada grep, algoritma Boyer-Moore juga menghemat waktu kita dalam melakukan pencarian. Faktanya, algoritma Boyer-Moore merupakan salah satu alasan cepatnya GNU grep mengekskusi instruksi yang diberikan. Hal ini dikarenakan algoritma Boyer-Moore yang berbeda dengan algoritma lain, yaitu teknik pencariannya. Teknik *looking-glass* dan teknik pergeseran pada algoritma Boyer-Moore merupakan pembeda yang jelas dan signifikan dalam permormansi pencarian *pattern* pada teks dibandingkan dengan algoritma lain, seperti Brute Force atau Knuth-Pratt-Morris. Dengan teknik yang memeriksa karakter dari sebelah kanan *pattern*, algoritma Boyer-Moore dapat mendapat informasi lebih banyak dan teknik pergeseran yang terbagi menjadi tiga kasus khusus membuat algoritma Boyer-Moore memeriksa karakter sedikit mungkin dengan mengabaikan kemungkinan pencarian yang sudah pasti salah. Selain itu, algoritma Boyer-Moore melakukan pra-proses dengan menciptakan tabel fungsi kemunculan terakhir dari setiap karakter pada *pattern*. Hal ini mempercepat algoritma Boyer-Moore dalam menentukan pergeseran *pattern* apabila terjadi *mismatch*. Namun, pada GNU grep tidak hanya menggunakan Boyer-Moore saja, terkadang ada beberapa *regular expressions* yang tidak dapat/ tidak efisien menggunakan algoritma Boyer-Moore. Ada beberapa ekspresi yang mengharuskan menggunakan algoritma yang mengecek *pattern* dari kiri seperti Brute Force, contohnya saat mencari dengan karakter *anchor*. Selain itu, algoritma Boyer-Moore juga tidak memungkinkan pencarian dimana terdapat meta-karakter "*" yang menandakan kemunculan 0 atau lebih kali karakter atau himpunan karakter.

V. UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada Tuhan YME atas karunia-Nya karena telah selesainya makalah ini. Tak lupa penulis mengucapkan terima kasih kepada Dr. Ir. Rinaldi Munir dan Dr. Nur Ulfa Maulidevi atas bimbingannya dalam memberikan tugas makalah ini serta ilmu yang diberikan mengenai strategi algoritma. Pihak-pihak yang terkait dalam

pembuatan makalah ini pun tak lupa penulis mengucapkan terima kasih atas bantuannya selama proses pengerjaan makalah ini. Akhir kata penulis juga mengucapkan terima kasih kepada rekan-rekan HMIF ITB (Himpunan Mahasiswa Informatika ITB) atas kesempatan dan kerjasamanya.

REFERENSI

[1] "The GNU Operating system". Diakses tanggal 18-08-2008.

[2] "new UNIX implmentation". Diambil pada 18-08-2008.

[3] Boyer R.S., Moore J.S., 1997, A fast string searching algorithm. *Communications of the ACM*.

[4] Stallman, Richard (March 9 2006). *The Free Software Movement and the Future of Freedom*. Zagreb, Croatia: FSF Europe.

[5] Yi Peng; Fu Li; Ali Mili (January 2007). "Modeling the evolution of operating systems: An empirical study". Journal of Systems and Software (Elsevier).

[6] Stallman, Richard (1986), "KTH", *Philosophy* (speech), GNU, Stockholm, Sweden: FSF.

[7] Munir, Rinaldi. 2007. Diktat Kuliah IF2211. Strategi Algoritma. Bandung: Institut Teknologi Bandung.

[8] Why GNU grep is fast [online] http://lists.freebsd.org/pipermail/freebsd-current/2010-August/019310.html

[9] Using Grep & Regular Expressions to Search for Text Ptterns in Linux [online] https://www.digitalocean.com/community/tutorials/using-grep-regular-expressions-to-search-for-text-patterns-in-linux

[10] Watson, B.W., 1995, *Txonomies and Toolkits of Regular Language Algorithms*, Ph. D. Thesis, Eindhoven University of Technology, The Netherlands.

[11] Gonnet, G.H., Beaza-Yates, R.A., 1991. *Handbook of Algorithms and Data Structures in Pascal and C*, 2nd Edition, Chapter 7, pp. 251-288, Addison-Wesley Publishing Company.

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, 7 Mei 2016

Muhammad Gumilang
13514092