Algortima Pencocokan String

Najib Darmawan S2213001NIM
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
najib.darmawan@gmail.com

Abstract
Pencocokan string merupakan persoalan yang sangat sering ditemukan pada bidang informatika. Implementasi algortima pencocokan string yang mangkus sangat dibutuhkan untuk mempercepat komputasi pencarian string. Algortima pencocokan string yang terkenal adalah Knuth-Morris-Pratt (KMP) dan Boyer-Moore. Kedua algortima tersebut sangat mangkus jika dibandingkan dengan algoritma Brute Force. Pada makalah ini, penulis mencoba mencoba menjelaskan algoritma pencocokan string yang penulis buat sendiri yang selanjutnya disebut algoritma Najib-Darmawan dan membandingkannya dengan algoritma Knuth-Morris-Pratt (KMP) dan Boyer-Moore.

***Index Terms*— Pencocokan String, Knuth-Morris-Pratt (KMP), Boyer-Moore.**

I. PENDAHULUAN

Dalam penggunaan komputasi sehari-hari salah satu fungsi dasar yang sering dibutuhkan adalah pencocokan string. Sudah tidak dapat dibantah lagi jika setiap aplikasi menggunakan fungsi tersebut. Mulai dari aplikasi social media, pengolah kata, web browser, bahkan game sekalipun . Pencarian string pada umumnya dapat berupa suku kata, kata, frasa, hingga kalimat.

Dengan kebergantungan terhadapat fungsi pencocokan string yang begitu tinggi, dibutuhkan algoritma yang mangkus dan dapat menenmukan string yang dicari jika memang ada. Oleh karena itu, penulis mencoba membuat algoritma pencocokan strign yang lebih baik dibandingkan brute force, Knuth-Morris-Pratt (KMP), dan Boyer-Moore.

II. DASAR TEORI

Algortima pencocokan string merupakan salah satu algoritma pencocokan pola. Pola tersebut dapat berupa teks ataupun citra. Pecncocokan string umumnya dilakukan dengan panjang pola (m) yang jauh lebih pendek jika dibandingkan dengan panjang teks (n). Terdapat 3 algoritma yang sering dijadikan dasar untuk mencocokkan string yaitu algoritma brute force, algoritma Knuth-Morris-Pratt (KMP), dan algortima Boyer-Moore.

2.1 Algoritma Knuth-Morris-Pratt (KMP)

Algoritma KMP mencocokan pattern dengan menelusuri text dari kiri ke kanan. Jika ditemukan ketidakcocokan pola saat mencocokkan, maka pergeseran dilakukan sebanyak jumlah terbesar prefik yang juga merupakan sufik dari pattern. Proses menentukan jumlah pergeseran tersebut dilakukan dengan fungsi pinggiran. Fungsi pinggiran melakukan pencocokan prefik dari pattern dengan pattern itu sendiri. Pada implementasi KMP proses fungsi pinggiran dilakukan sebelum mencocokan pattern dengan text.

Kompleksitas waktu yang dibutuhkan algoritma KMP untuk menghitung fungsi pinggiran sebanyak O(m) dengan m adalah panjang pattern. Selain itu, kompleksitas waktu yang dibuthkan untuk proses pencocokan string sebanyak O(n). Sehingga, kompleksitas waktu algoritma KMP adalah O(m+n).

Berikut adalah algoritma KMP dalam bahasa Java:

a. Fungsi Pinggiran

```
public static int[]computeFail(String
pattern){
  int m = pattern.length();
  int j = 0;
  int i = 1;

  int fail[] = new int[m];
  fail[0] = 0;
 while(i < m){
  if(pattern.charAt(j) ==
  pattern.charAt(i)){
     fail[i] = j + 1;
     i++; j++;
  }
  else if(j > 0) j = fail[j-1];
  else{
     fail[i] = 0;
     i++;
  }
 }
 return fail;
}
```

b. Algoritma KMP

```
public static int kmpMatch( String
pattern,String text)
{
  int n = text.length();
  int m = pattern.length();

  int fail[]=computeFail(pattern);

  int i = 0, j = 0;

  while(i < n){
   if(pattern.charAt(j) ==
   text.charAt(i)){
        if(j == m -1){
          return i - m + 1;
        }
        i++; j++;
   }
   else if(j > 0){
        j = fail[j-1];
   }
   else { i++; }
  }
  return -1;
}
```

2.2 Algoritma Boyer-Moore

Algoritma Booyer-Moore didasarkan pada 2 teknik:

1. Teknik looking-glass

   Mencocokan pattern dalam text dimulai dari indeks terakhir pattern dan indek ke-i text dimana i adalah panjang pettern.

2. Teknik character-jump

   Ketika terjadi ketidakcocokan, geser indeks text dan pattern. Terdapat 3 kemungkinan pergeseran indeks text dan pattern:

   a. Jika karakter pada indeks terjadinya ketidakcocokan ada pada pattern, maka geser indeks pattern tersebut ke indeks terjadinya ketidakcocokan dengan indeks text digeser sebanyak pergeseran indeks pattern.

   b. Jika karakter pada indeks terjadinya ketidakcocokan ada pada pattern tetapi penggeseran indeks pattern ke indeks terjadinya ketidakcocokan tidak mungkin, maka geser indeks pattern sebanyak 1 karakter dengan indeks text digeser 1 karakter.

   c. Jika karakter pada indeks terjadinya ketidakcocokan tidak ada pada pattern, maka geser indeks pattern ke indeks pertama dengan indeks teks digeser 1 karakter.

Pada implementasinya teknik character-jump dijadikan fungsi *last occurrence*. Pada fungsi tersebut setiap alfabet dicatat indeks terakhir kemunclannya, jika alfabet tidak ada pada pattern maka nilainya -1.

Algoritma Boyer-Moore akan efektif jika jumlah alfabet pada teks besar, tidak efektif saat jumlah alfabet kecil. Kompleksitas waktu terburuk algortima Boyer-Moore:

1. O(A) untuk mencatat nilai *last occurrence* setiap alfabet.

2. O(nm) untuk mencocokan pattern dengan text

Jadi kompleksitas terburuk algoritma Boyer-Moore adalah O(nm+A).

Algortima Boyer-Moore dalam bahasa Java:

a. Fungsi kemunculan terakhir

```
public static int[]
buildLast(String pattern){
   int last[] = new int[65355];
   for(int i = 0; i < 65355;
   i++){
       last[i] = -1;
   }
   for(int i = 0; i <
   pattern.length(); i++){
       last[pattern.charAt(i)]=i;
   }
   return last;
}
```

b. Algoritma Boyer-Moore

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

   if(i > n - 1) {
     return -1;
   }
   int j = m - 1;
   do{
       if(pattern.charAt(j) ==
       text.charAt(i)){
         if(j == 0){
            return i;
         }
         else{
            i--; j--;}
```

```
            }
          else{
                int lo =
                last[text.charAt(i);
                i = i + m -
                Math.min(j, 1+lo);
                j = m - 1;
          }
      } while (i <= n - 1);
      return -1;
  }
```

III. ANALISIS PEMECAHAN MASALAH

Algoritma Najib-Darmawan merupakan algoritma pencocokan pola, khususnya string dengan didasarkan pada kemunculan karakter pertama dan terakhir string serta panjang string. Hal ini dilakukan karena jarang ditemukan kata yang memiliki karakter pertama dan terakhir yang sama dengan panjang yang sama.

Pencarian karakter pertama dan terakhir dilakukan tersendiri dengan menggunakan fungsi *find index*. Berikut adalah algoritma fungsi *find index* dalam bahasa Java:

```
public static int findIndex(char
f, char l, String text,
List<Integer> charIndex){
   int splitIndex = -1;
   for(int i=0 ;i<text.length();
   i++){
       if(f==text.charAt(i)){
          splitIndex++;
          charIndex.add(0,i);
       }
      if(l == text.charAt(i)){
          charIndex.add(charIndex.
          size(),i);
       }
   }
   return splitIndex;
 }
```

Variabel f dan l berturut-turut adalah karakter pertama dan karakter terakhir. Keduanya tidak boleh ditukar karena akan menyebabkan string yang dicari tidak ditemukan. Saat karakter ditemukan, indeks penemuan karakter diinput ke list pada elemen pertama untuk karakter pertama dan elemen terakhir untuk karakter terakhir. Keluaran dari fungsi tersebut adalah sebuah list integer yang merepresentasikan indeks kemunculan karakter pertama dan terakhir dan keluaran integer indeks pemisah antara indeks yang ditujukan untuk karakter pertama dan terakhir. List yang digunakakn untuk menyimpan kemunculan indeks hanya satu, hal ini betujuan untuk menghemat alokasi (tidak dua kali melakukan alokasi list).

Setelah didapatkan list integer dan integer indeks maka dapat dilakukan pencarian string dengan menggunakan algoritma Najib-Darmawan. Pada algoritma ini pencocokan dilakukan dengan membandingkan selisih kemunculan karakter pertama dan karakter terakhir pada teks yang dicari dengan panjang string yang dicari. Jika selisih tersebut sama dengan panjang pola yang dicari maka dilakukan pengecekan karakter ke m-1 sampai karakter kedua string. Jika sampai karakter kedua masih cocok maka string ditemukan. Berikut adalah algoritma Najib-Darmawan dalam bahasa Java:

```
public static int NDmatch(String
pattern, String text){
  int m = pattern.length();
  List<Integer> charIndex = new
  ArrayList<>();

  int x = findIndex
  (pattern.charAt(0),pattern.char
  At(m-1),text,charIndex);
  if(charIndex.isEmpty()){
    return -1;
  }
  else{
   int i = x+1, j = x;
   while(i < charIndex.size() &&
   j >= 0){
    if(charIndex.get(i) -
     charIndex.get(j) < m -1){
       i++;
    }
    else if(charIndex.get(i) -
     charIndex.get(j) == m - 1){
       int l=charIndex.get(i)-1;
       int k=m-2;
       while(k > 0 &&
       (pattern.charAt(k) ==
       text.charAt(l))){
         k--; l--;
       }
       if(k == 0) {
         return charIndex.get(j);
       }
       else { i++; }
    }
    else {
      if(j > 0){ j--; }
      else{ return -1; }
    }
   }
  }
  return -1;
}
```

Variabel i dan j berturut-turut adalah indeks untuk karakter terakhir dan karakter pertama. Ketika selisih lebih kecil dari panjang string, pindah ke indeks karakter terakhir berikutnya. Ketika selisih lebih besar, pindah ke indeks karakter pertama berikutnya. Ketika selisih sama, cek karakter teks pada indeks ke i-1 sampai karakter kedua string yang dicari. Ini dilakukan karena karakter

pertama dan terakhir sudah pasti sama. Keluaran dari fungsi ini adalah indeks ditemukannya string yang dicari.

Jika a adalah jumlah karakter pertama, b adalah jumlah karakter terakhir, n adalah panjang teks dan m adalah panjang string, maka kompleksitas waktu yang dibutuhkan untuk menemukan string:

- Mendapatkan indeks dari karakter pertama dan terakhir adalah O(n).
- Mencocokan string dalam kasus terburuk O(max(a,b)*(m-2))
- Mencocokan string dalam kasus terbaik O(min(a,b)*(m-2)) dengan asumsi kasus terbaik tidak termasuk ketika string tidak ditemukan karena tidak ada karakter pertama dan terakhir pada teks (O(n)).

Jadi kompleksitas waktu yang dibutuhkan untuk algoritma Najib-Darmawan dalam kasus terburuk adalah O(n+(max(a,b)*(m-2))) dan dalam kasus terbaik O(n+(min(a,b)*(m-2))).

IV. HASIL PENGUJIAN

Dilakukan beberapa kali pengujian pada algoritma Najib-Darmawan. Berikut adalah hasil pengujian dan perbandingan dengan menggunakan teks pada file Template Makalah IF2211.docx. Jumlah kata pada dokumen tersbut adalah 2145 dan jika dijadika string maka panjangnya 13157.

a. Pencarian pada bagian awal dokumen

1. String pencarian `key words or phrases`.

	KMP	BM	ND
Perbandingan pada fungsi	19	65375	13157
Perbandingan pada algoritma	824	84	63
Total	843	65459	13220

2. String pencarian `Jl.`

	KMP	BM	ND
Perbandingan pada fungsi	2	65358	13157
Perbandingan pada algoritma	179	63	2
Total	181	65421	13159

3. String pencarian `This document is a template for Microsoft Word versions 6.0 or later.`

	KMP	BM	ND
Perbandingan pada fungsi	68	65424	13157
Perbandingan pada algoritma	1217	123	93
Total	1285	65547	13250

b. Pencarian pada bagian tengah dokumen

1. String pencarian ` this document are`

	KMP	BM	ND
Perbandingan pada fungsi	17	65372	13157
Perbandingan pada algoritma	6992	831	1160
Total	7009	66203	14317

2. String pencarian `“zero-`

	KMP	BM	ND
Perbandingan pada fungsi	5	65361	13157
Perbandingan pada algoritma	8293	1593	56
Total	8298	66954	13213

3. String pencarian `Be sure that the symbols in your equation have been defined before the equation appears or immediately following. Italicize symbols (T might refer to temperature, but T is the unit tesla). Refer to “(1),” not “Eq. (1)” or “equation (1),” except at the beginning of a sentence: “Equation (1) is ... .`

	KMP	BM	ND
Perbandingan pada fungsi	298	65654	13157
Perbandingan pada algoritma	8160	463	439
Total	8456	66117	13596

c. Pencarian pada bagian akhir dokumen

1. String pencarian ` Bandung, 29 April 2010 `

	KMP	BM	ND

Perbandingan pada fungsi	24	65380	13157
Perbandingan pada algoritma	13151	754	2105
Total	13175	66134	15262

2. String pencarian `PERNYATAAN`

	KMP	BM	ND
Perbandingan pada fungsi	9	65365	13157
Perbandingan pada algoritma	12979	1318	72
Total	12988	66683	13229

3. String pencaria `C. J. Kaufman, Rocky Mountain Research Lab., Boulder, CO, private communication, May 1995.`

	KMP	BM	ND
Perbandingan pada fungsi	90	65445	13157
Perbandingan pada algoritma	12947	521	360
Total	13037	65966	13517

Rata-rata perbandingan ketiga algortima akan disajikan pada tabel berikut ini:

	KMP	BM	ND
Perbandingan pada fungsi	59,11	65414,88	13157
Perbandingan pada algoritma	7193,56	638,89	483,33
Total	7249,44	66053,77	13640,33

Selisih rata-rata ketiga algoritma terhadap yang terbaik setiap bagiannya

	KMP	BM	ND
Perbandingan pada fungsi	0	65355,77	13097,89
Perbandingan pada algoritma	6710,23	155,56	0
Total	0	58804,33	6391.3

V. ANALISIS

Pada pencarian bagian awal dokumen algortima KMP selalu lebih baik pada perbandingan fungsi dan total perbandingan dan algoritma ND lebih baik dalam mencocokan string. Algoritma Boyer-Moore tidak satupun lebih baik dibandingkan dua algoritma yang lain. Hal ini dikarenakan penggunaan unicode pada fungsi *last occurrence* . Seperti diketahui bersama karakter uni code jika direpresentasika ke dalam kode biner ada sebanyak 65355. Jika menggunakan standar yang lain maka tidak semua karakter dapat dikenali. Pada pencarian string 'Jl.' Algoritma ND sangat sedikit melakukan pencocokan string, hal ini dikarenakan karakter 'J' jarang muncul dan panjang string yang dicari pendek.

Pada pencarian bagian tengah dokumen algoritma KMP lebih keadaan masih sama, hanya saja algoritma Boyer-Moore sekali lebih baik saat mencocokan string dibandingkan algortima ND. Hal ini dikarenakan karakter awal yaitu ' ' atau spasi sangat sering muncul. Selain itu juga algortima Boyer-Moore hampir sama dengan algortima ND pada pengujian b.3. hal ini dikarenakan karakter terakhir string yang dicari sering muncul.

Pada pencarian bagian ahir dokumen algortima ND relatif mendekati hasil yang diperoleh algortima KMP. Hal ini dikarenakan algortima KMP sangat bergantung pada indeks ditemukannya string.

Pada tabel selisih rata-rata, jika dibandingkan antara selisih rata-rata total algoritma ND dan selisih rata-rata perbandingan pencocokan string, algoritma ND lebih baik. Hal ini menunjukkan bahwa algoritma ND mungkin saja dapat lebih baik dari algoritma KMP jika dapat mengoptimalkan fungsi *find index*.

VI. KESIMPULAN

Kesimpulan yang dapat diambil dari bab analisis adalah sebagai berikut:

1. Algoritma Najib-Darmawan lebih baik dibandingkan algoritma Boyer-Moore tetapi lebih jelek dibandingkan algortima Knuth-Morris-Pratt.

2. Algoritma Najib-Darmawan akan menghasilkan solusi yang baik apabila karakter pertama dan karakter terakhir string yang dicari tidak sering muncul.

3. Algortima Najib-Darmawan sangat baik dalam melakukan pencocokan string tetapi tidak pada fungsi *find index*.

4. Algoritma Najib-Darmawan akan lebih baik jika dapat mengoptimalkan fungsi *find index*.

Saran untuk pengembangan selanjutnya:

1. Optimalkan fungsi *find index*.

2. Optimalisasi dapat dilakukan dengan tidak mencari kemunculuan karakter langsung pada seluruh teks, tetapi dengan batasan tertentu.

REFERENSI

[1] Slide kuliah IF2211 Startegi Algoritma Pencocokan String(2015).ppt

[2] http://cs.stackexchange.com/questions/19564/complexity-analysis-of-while-loop-with-two-conditions (diakses pada 4 mei 2015 pukul 14.25)

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, 4 Mei 2015

Najib Darmawan
S2213001