PENERAPAN *STRING MATCHING* DENGAN ALGORITMA *BOYER MOORE* PADA APLIKASI *FONT ITALIC* UNTUK DETEKSI KATA ASING

Rohmat Indra Borman [1)], Agus Pratama [2)]

[1)] Komputerisasi Akuntansi, STMIK Teknokrat
[2)] Teknik Informatika, STMIK Teknokrat
Jl. H.ZA Pagaralam, No 9-11, Labuhanratu,Bandarlampung
Email : rohmat_indra@teknokrat.ac.id[1)], aguspratama@gmail.com [2)]

Abstrak

Dalam karya ilmiah penulisan untuk kata asing, seperti kata berbahasa inggris, berbahasa yunani dan bahasa lainya dibuat dengan memiringkan kata tersebut. Aplikasi pengolah kata yang biasanya digunakan adalah Microsoft Office Word. Untuk memiringkan kata asing pada Microsoft Office Word menggunakan salah satu tools yang ada di toolbar Microsoft Office Word yaitu italic. Algoritma boyer moore merupakan salah satu algoritma yang digunakan untuk melakukan pencocokan string (string matching). Algoritma ini merupakan jenis exact string matching algorithm yang melakukan pencocokan string secara tepat dengan susunan karakter dalam string yang dicocokkan memiliki jumlah maupun urutan karakter dalam string yang sama. Cara kerja algoritma ini adalah dengan melakukan pencocokan dari kanan ke kiri yaitu men-scan karakter pattern dari kanan ke kiri dimulai dari karakter paling kanan. Penerapan algoritma boyer moore pada aplikasi font italic, akan mencari semua kemungkinan kata asing di dalam dokumen microsoft office word dengan yang ada di database aplikasi untuk membuat otomatis tercetak miring.

Kata kunci: *Aplikasi, font italic, katas asing, algoritma, string matching, boyer moore, exact string matching.*

1. Pendahuluan

Dalam sebuah karya ilmiah banyak menggunakan kata-kata asing khususnya kata berbahasa inggris. Penulisan karya ilmiah tersebut biasanya menggunakan aplikasi pengolah kata diantaraya *Microsoft Office Word*. Pada *Microsoft Office Word* untuk membuat kata tercetak miring harus menggunakan *tools* yang ada di *toolbar yaitu italic*. Hal ini berakibat memperlambat waktu dalam penulisan jika harus mengubah satu per satu kata-kata berbahasa inggris tersebut menjadi cetak miring dan kemungkinan kata asing tersebut terlewatkan untuk membuat tercetak miring. Selain itu akan cukup melelahkan apabila penulis karya ilmiah tersebut harus berulang-ulang menemukan kesalahan penulisan kata berbahasa inggris. Untuk menyelesaikan masalah diatas dibutuhkan sebuah aplikasi yang berguna untuk mencetak miring sebuah kata berbahasa inggris secara otomatis. Beberapa algoritma dapat digunakan untuk mendeteksi kata bahasa inggris yang akan di cetak miring.

Algoritma *boyer moore* merupakan algoritma yang digunakan untuk melakukan pencocokan *string*. Algoritma ini merupakan jenis *exact string matching algorithm* yang merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. Algoritma ini melakukan pencocokan dari kanan ke kiri yaitu men-*scan* karakter *pattern* dari kanan ke kiri dimulai dari karakter paling kanan.

2. Pembahasan

2.1. *String Matching*

Pencocokan *string* atau *string matching* merupakan bagian penting dari sebuah proses pencarian *string (string searching)* dalam sebuah dokumen (Saragih, 2013). Hasil dari pencarian sebuah *string* dalam dokumen tergantung dari teknik atau cara pencocokan *string* yang digunakan. Pencocokan *string (string matching)* secara garis besar dapat dibedakan menjadi dua yaitu sebagai berikut :

a. *Exact string matching*, merupakan pencocokan *string* secara tepat dengan susunan karakter dalam *string* yang dicocokkan memiliki jumlah maupun urutan karakter dalam *string* yang sama. Contoh : kata *open* akan menunjukkan kecocokan hanya dengan kata *open.*
b. *Inexact string matching* atau *Fuzzy string matching*, merupakan pencocokan *string* secara samar, maksudnya pencocokan *string* dimana *string* yang dicocokkan memiliki kemiripan dimana keduanya memiliki susunan karakter yang berbeda (mungkin jumlah atau urutannya) tetapi *string-string* tersebut memiliki kemiripan baik kemiripan tekstual/penulisan *(approximate string matching)* atau kemiripan ucapan *(phonetic string matching)*

2.2. Algoritma *Boyer Moore*

Ide utama dari algoritma *boyer moore* adalah dengan melakukan pencocokan dari paling kanan *string* yang dicari. Dengan menggunakan algoritma ini, secara rata-rata proses pencarian akan lebih cepat dibandingkan dengan proses pencarian lainnya. Ide dibalik algoritma

ini adalah bahwa dengan memulai pencocokkan karakter dari kanan, dan bukan dari kiri, maka akan lebih banyak informasi yang didapat (Efendi, 2012).

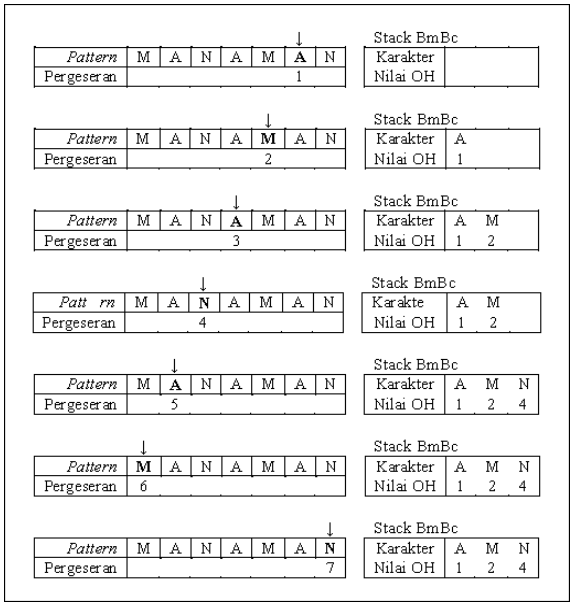
Algoritma *boyer moore* menggunakan metode pencocokan *string* dari kanan ke kiri yaitu men-*scan* karakter *pattern* dari kanan ke kiri dimulai dari karakter paling kanan. Algoritma *Boyer Moore* menggunakan dua fungsi *shift* yaitu *good-suffix shift* dan *bad-character shift* untuk mengambil langkah berikutnya setelah terjadi ketidakcocokan antara karakter *pattern* dan karakter teks yang dicocokkan.

2.3. Cara Kerja Algoritma Boyer Moore

Cara kerja dari algoritma *Boyer Moore* adalah sebagai berikut :

a) Menjalankan prosedur preBmBc dan preBmGs untuk mendapatkan inisialisasi.

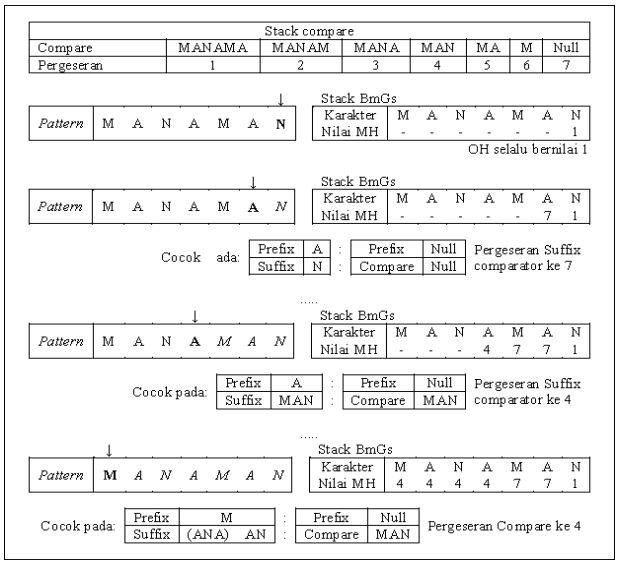
1. Menjalankan prosedur preBmBc.
   Fungsi dari prosedur ini adalah untuk menentukan berapa besar pergeseran yang dibutuhkan untuk mencapai karakter tertentu pada *pattern* dari karakter *pattern* terakhir/terkanan. Hasil dari prosedur preBmBc disimpan pada tabel BmBc.
   Contoh kasus :
   Pattern : MANAMAN



Gambar 1. Penyelesaian contoh kasus prosedur PreBmBc

2. Menjalankan prosedur preBmGs.
   Sebelum menjalankan isi prosedur ini, prosedur *suffix* dijalankan terlebih dulu pada *pattern*. Fungsi dari prosedur *suffix* adalah memeriksa kecocokan sejumlah karakter yang dimulai dari karakter terakhir/terkanan dengan sejumlah karakter yang dimulai dari setiap karakter yang lebih kiri dari karakter terkanan tadi. Hasil dari prosedur *suffix* disimpan pada tabel *suff*. Jadi *suff*[i] mencatat panjang dari *suffix* yang cocok dengan segmen dari *pattern* yang diakhiri karakter ke-i.

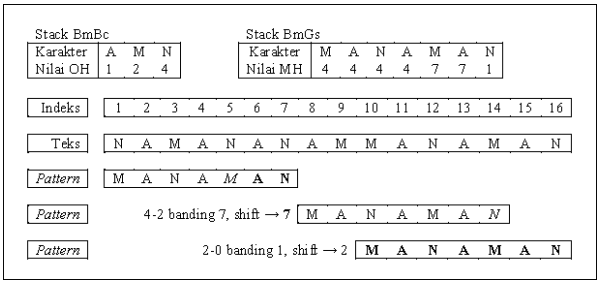
   Contoh kasus :
   Pattern : MANAMAN



Gambar 2. Penyelesaian contoh kasus prosedur preBmGs

3. Dengan prosedur preBmGs, dapat diketahui berapa banyak langkah pada *pattern* dari sebeuah segmen ke segmen lain yang sama yang letaknya lebih kiri dengan karakter di sebelah kiri segmen yang berbeda. Prosedur preBmGs menggunakan tabel *suff* untuk mengetahui semua pasangan segmen yang sama.

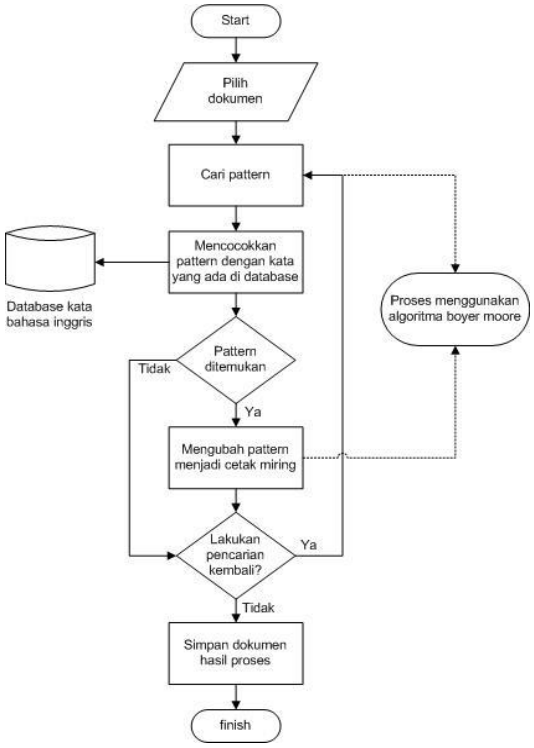
b) Dilakukan proses pencarian *string* dengan menggunakan hasil dari prosedur preBmBc dan preBmGs yaitu tabel BmBc dan BmGs.
   Contoh kasus :
   Pattern : MANAMAN
   Teks : NAMANANAMMANAMAN



Gambar 3. Penyelesaian contoh kasus prosedur BM

2.4. Gambar Alur Sistem

Berikut rancangan alur cara kerja sistem berupa *flowchart* yang digunakan oleh penulis dalam melakukan penelitian:

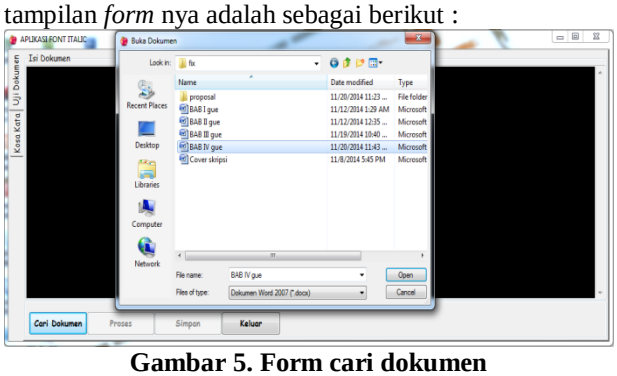


Gambar 4. Flowchart Alur Sistem

Dari *flowchart* diatas dapat dijelaskan dari awal proses peng*input*an dokumen, proses pencarian kata bahasa inggris didalam dokumen tersebut sampai dengan hasil cetak miring kata bahasa inggris yang kemudian disimpan menjadi sebuah dokumen baru tanpa mengubah dokumen aslinya.

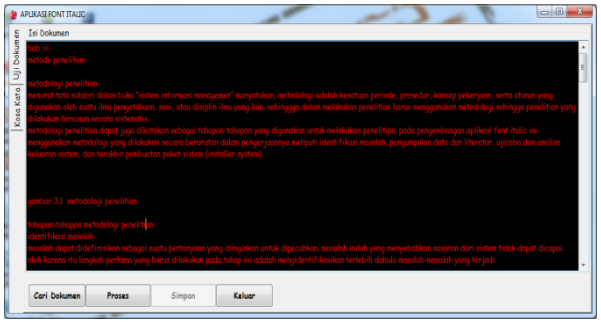
2.5. Cara Kerja Aplikasi dan Pengujian

Setelah aplikasi dijalankan, maka akan tampil *form* uji dokumen yang berisi tombol cari dokumen, proses, simpan, keluar dan sebuah memo untuk menampilkan isi dokumen. Untuk meng*input*kan dokumen ke dalam aplikasi dapat menggunakan tombol cari dokumen, maka tampilan *form* nya adalah sebagai berikut :



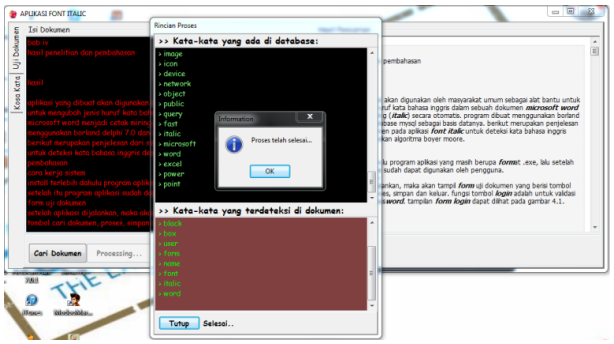
Gambar 5. Form cari dokumen

Setelah dokumen yang akan diproses di*input*kan maka tampilan *form* setelah dokumen di*input*kan adalah sebagai berikut :



Gambar 6. Form buka dokumen

Ketika ditekan tombol proses maka aplikasi akan mulai pencarian kata. Tampilan *form* proses adalah sebagai berikut :



Gambar 7. Form proses

Dari proses tersebut yang dilakukan pertama kali adalah menjalankan prosedur preBmBc dan preBmGs untuk mendapatkan inisialisasi.

```
for a:=0 to StrLen(Pchar(sumber)+20) do
  begin
    if(index<=StrLen(Pchar(sumber)))then
    begin
    teks1 := Copy(sumber,index,panjangPattern);
    teks1 := LowerCase(teks1);
    c:=panjangPattern;
    if(LowerCase(teks1)=LowerCase(teks2))then
      begin
        index := index+1;
        jumlah:=jumlah+1;
        if(Pos(teks1,Form1.Memo1.Text)<1)then
          Form1.Memo1.Lines.Add(' > '+teks1);
      end
```

Gambar 8. Menentukan Pegeseran

Proses diatas untuk menentukan berapa besar pergeseran yang dibutuhkan untuk mencapai karakter tertentu pada pattern dari karakter pattern terakhir/terkanan. Kemudian dilakukan pencarian

```
for b:=1 to panjangPattern do
  begin
    if((index<=StrLen(Pchar(sumber))) and (c>0))then
      begin
        if(LowerCase(teks1[c])=LowerCase(teks2[c]))then
          begin
            index := index+(panjangPattern-c);
          end
        else
          begin
            posisi  := Pos(teks1[c],teks2);
              Form1.Memo2.Lines.Add('Posisi Karakter '+IntToStr(posisi));
            if(posisi>0)then
              begin
                index := index+(panjangPattern-posisi);
                c:=0;
              end
            else
              begin
                c:=0;
                index := index+panjangPattern;
              end;
          end;
          Form1.Memo2.Lines.Add('INDEX '+IntToStr(index+1));
      end;

    c:=c-1;
  end;
end;
```

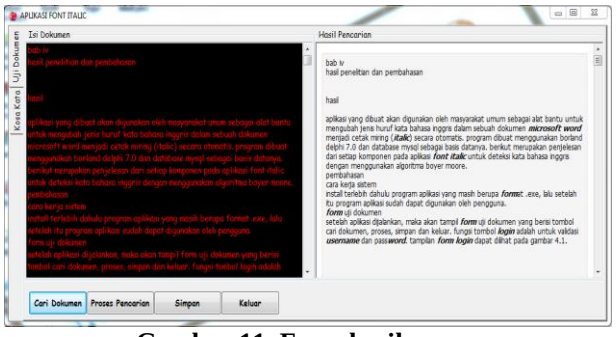
Gambar 9. Memeriksa Kecocokan Karakter

Setelah itu dilakukan pemeriksaan terhadap kecocokan sejumlah karakter yang dimulai dari karakter terakhir/terkanan dengan sejumlah karakter yang dimulai dari setiap karakter yang lebih kiri dari karakter terkanan tadi. Kemudian dilakukan pencarian terhadap hasil pencocokan.

```
                    index := index+(panjangPattern-posisi);
                    c:=0;
                  end
                else
                  begin
                    c:=0;
                    index := index+panjangPattern;
                  end;
              end;
//            Form1.Memo2.Lines.Add('INDEX '+IntToStr(index+1));
          end;

          c:=c-1;
        end;
    end;
//  Form1.Memo2.Lines.Add('-----------------------------------------------
  end;

hasil := LowerCase(hasil);
hasil := StringReplace(hasil,teks2,'<i><b>'+teks2+'</b></i>',[rfReplaceAll]);

hasil2 := LowerCase(hasil2);
hasil2 := StringReplace(hasil2,teks2,'<b>'+teks2,[rfReplaceAll]);
```

Gambar 10. Menentukan Hasil Pencocokan

Setelah proses pencarian selesai, maka akan didapatnya hasil proses. Tampilan *form* hasil proses pencarian adalah sebagai berikut :



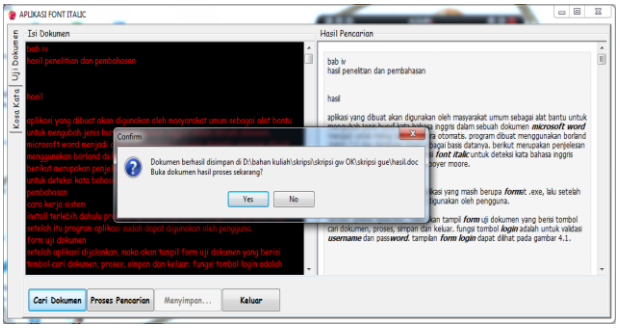
Gambar 11. Form hasil proses

Dari proses pencarian kata asing tersebut dilakukan pengujian proses per untuk deteksi kata asing dengan menerapkan algoritma *boyer moore* menghasilkan tabel berikut ini :

Tabel 1. Hasi Pengujian Pencocokan String

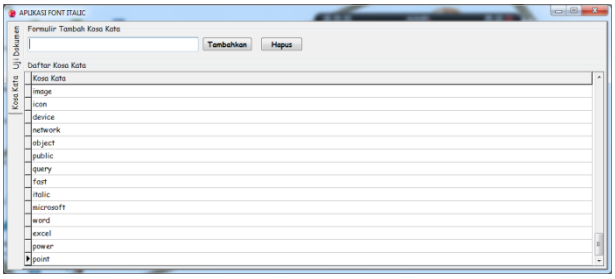
Halaman (Lembar)	Jumlah Kata	Ukuran *File*	Waktu (Detik)	Jumlah kata yang tercetak miring	Jumlah kata di database
1	184	16 Kb	2,23	17	125
2	368	17 Kb	4,48	34	125
4	736	19 Kb	11,94	68	125
8	1472	22 Kb	39,60	136	125
16	2944	29 Kb	151,86	272	125
32	5888	42 Kb	348,23	544	125
64	11.776	63 Kb	587,86	1088	125

Setelah dilakukan proses deteksi kata asing dapat dilakukan penyimpanan file yang hasilnya berupa dokumen *microsoft word* yang baru.



Gambar 12. Form simpan

Untuk menambah kata pada *database* pengguna dapat menambah kata bahasa inggiris pengguna system dapat menambahnya pada menu kosa kata :



Gambar 13. Form kosa kata

3. Kesimpulan

Berdasarkan penelitian yang telah dilakukan dapat disimpulkan :

1. Algoritma *boyer moore* dapat digunakan sebagai pencocokan *string* untuk pencarian kata berbahasa inggris dengan mencari kata yang sama dalam database melalui proses pencocokan dari kanan ke

kiri yaitu men-*scan* karakter *pattern* dari kanan ke kiri dimulai dari karakter paling kanan.

2. Pencocokan kata berbahasa inggris dapat diterapkan pada pengolah kata *Microsoft Office Word* dan dapat disimpan melalui format *Microsoft Office Word* yang sama dengan kata berbahasa inggris yang sudah tercetak miring.
3. Berdasarkan hasil pengujian proses deteksi kata bahasa inggris dengan algorima *boyer moore* jumlah katasing yang terdeteksi sudah sesuai dengan kata yang ada dalam database, tetapi semakin banyak jumlah kata yang ada pada teks maka semakin bertambah waktu yang dibutuhkan dalam melakukan pencarian.

Daftar Pustaka

[1] Charras, Christian., Lecroq, Thierry., *Handbook of Exact String-Matching Algotithms*. Oxford Unifersity Press, 1997.

[2] Effendi, Diana., Kurnaedi, Andri. Pengembangan Algoritma Boyer Moore pada Translator Bahasa Pemrograman, FTIK, Universitas Komputer Indonesia, 2012.

[3] Minandar, Arie., Tanoto, Andri., Tanadi, Davis. Aplikasi Algoritma Pencarian String Boyer-Moore Pada Pencocokan DNA. Departemen Teknik Informatika, Institut Teknologi

[4] Sagita, Vina., Irmina, Maria. Studi Perbandingan Implementasi Algoritma Boyer-Moore, Turbo Boyer-Moore dan Tuned Boyer-Moore dalam Pencarian String. Jurusan Teknik Informatika, Universitas Multimedia Nusantara, Tangerang, 2013.

[5] Saragih, May Aprina. 2013. Implementasi Algoritma Brute Force dalam Pencocokan Teks Font Italic untuk Kata Berbahasa Inggris pada Dokumen Microsoft Word. Jurusan Teknik Informatika, STIMIK Budidarma Medan