Perbandingan Algoritma String Matching yang Digunakan dalam Pencarian pada Search Engine

Eldwin Christian / 13512002
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
eldwin.christian@students.itb.ac.id

***Abstract*—String Matching adalah suatu algoritma yang digunakan untuk mencari suatu string (pattern) apakah terdapat dalam suatu string lainnya (teks). Pencarian ini dilakukan dengan cara mencocokkan karakter yang ada pada pattern dengan karakter pada teks yang ingin dicari. Dalam pencocokkan karakter antara pattern dan teks terdapat beberapa algoritma yang dapat digunakan, yaitu Algoritma Pola Tunggal, Algoritma yang menggunakan himpunan terhingga pola, dan algoritma yang menggunakan tak terhingga pola.**

Paper ini akan membahas tentang penggunaan dan perbandingan hasil dari algoritma dalam string matching yang dapat digunakan untuk pencarian pada mesin pencari agar dapat menghasilkan hasil yang optimal untuk pencarian yang sama.

***Index Terms*— Mesin Pencari, String Matching, Mesin Pencari, Boyer Moore, Brute Force, Knuth-Morris-Pratt.**

I. PENDAHULUAN

Pada zaman sekarang ini, penyebaran informasi telah berjalan dengan cepat. Dengan menggunakan internet, kita dapat dengan mudahnya mencari informasi yang kita butuhkan sehingga kita tidak perlu repot-repot untuk mencari buku yang berisi tentang informasi yang kita inginkan agar dapat mengetahui hal tersebut. Berbagai macam orang dari berbagai umur dan kalangan akan dengan mudahnya memasukkan "kata kunci" dari informasi yang ingin dicarinya pada mesin pencari untuk mendapatkan informasi mengenai hal tersebut melalui internet.

Mesin pencari adalah suatu sistem perangkat lunak yang dijalankan pada *web browser* untuk mencari informasi melalui internet pada *World Wide Web* yang tersimpan dalam ftp, www, news group ataupun publikasi milis dalam sejumlah komputer *client*. Mesin pencari ini memerlukan "kata kunci" dari pengguna untuk menampilkan informasi yang berhubungan dengan kata kunci tersebut. Hasil pencarian akan didapatkan dari dokumen – dokumen yang tersedia pada komputer *server*. Tampilan hasil pencarian umumnya ditampilkan dalam bentuk daftar yang diurutkan berdasarkan tingkat akurasi atau banyaknya pengunjung pada berkas dokumen yang biasa disebut *hits* Hasil dari pencarian yang ditampilkan tersebut dapat berupa *website* lain, gambar, atau informasi lainnya yang berhubungan dengan "kata kunci" yang tersebut. Ada juga mesin pencari yang melakukan pencarian terlebih dahulu pada internet untuk kemudian menyimpan hasil pencarian tersebut pada basis data.

Cara kerja mesin pencari dapat dibagi menjadi 3 bagian besar, yaitu *web crawling*, *indexing*, dan *searching*. Pada tahap *web crawling*, mesin pencari akan menjelajahi banyak halaman web yang diambil dengan menggunakan *web crawler*. *Web crawler* akan mengambil setiap pranala(link) yang ada pada suatu web yang sedang dijelajahinya dan kemudian juga akan melakukan tahap *web crawling* untuk pranala tersebut. Pada tahap *indexing*, isi dari setiap halaman web yang dijelajahi akan dianalisis dan diberikan indeks. Pemberian indeks ini dapat ditentukan berdasarkan isi yang diambil, mungkin berupa judul, subjudul atau bagian khusus dari isi halaman. Pada tahap *searching*, mesin pencari akan menerima "kata kunci" dari pengguna yang berupa *query*, kemudian mesin pencari akan mencari indeks dari kata yang terdapat pada *query* tersebut. Pencarian indeks ini dapat dilakukan dengan banyak cara pula, seperti membagi *query* menjadi beberapa *substring* lalu melakukan pencocokan atau melakukan pencarian indeks per kata dari *query*. Setelah pencarian indeks untuk *query* selesai dilakukan, mesin pencari kemudian memberikan daftar halaman web atau dokumen yang sesuai dengan kriteria pencarian. Hasil ini biasanya disertai dengan ringkasan singkat mengenai halaman web yang dirujuk oleh hasil pencarian.

Manfaat mesin pencari tergantung dengan hubungan dari hasil – hasil yang diberikannya. Pada kenyataan, ada kemungkinan yang sangat besar bahwa kata/frasa yang ada pada *query* dari pengguna terdapat pada begitu banyak halaman web, sehingga tentunya harus ada algoritma penentuan "peringkat" oleh mesin pencari untuk tiap halaman yang yang akan ditampilkan sebagai hasil pencarian. Sehingga halaman web yang ditampilkan sebagai hasil pencarian pengguna akan berupa halaman yang paling sering dikunjungi, paling populer, atau lebih relevan terhadap *query* yang sedang dicari.

II. TEORI DASAR

A. Algoritma Brute Force

Algoritma *Brute Force* adalah pencarian yang dilakukan untuk membandingkan *pattern* P dengan teks T untuk setiap kemungkinan yang ada pada perpindahan *pattern* sebanyak 1 langkah hingga suatu *pattern* yang dicari cocok dengan *substring* dari teks atau hingga semua tempat / kemungkinan yang ada telah dicoba.

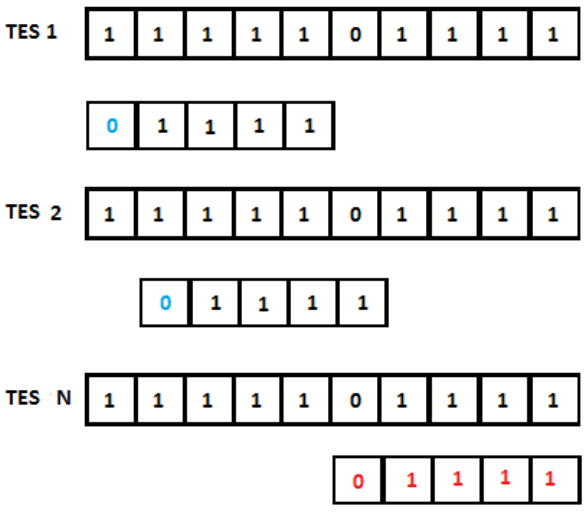
```
function match(String text, String pattern) => boolean
{
   for (int i← start of string until i<= text.length-
pattern.length)
        {
             Int j←0;
             while (j < pattern.length and pattern[j] ==
text[i+j])
              j++;
             if (j == pattern.length())

                return true;
        }
   return false;
}
```

Pseudocode 1. Algoritma *Brute Force*

Beberapa kasus yang dapat terjadi saat melakukan pencarian dengan menggunakan algoritma *brute force* pada suatu teks dengan panjang n dan *pattern* dengan panjang m :

1. Kasus terbaik saat *pattern* tidak ditemukan, maka :

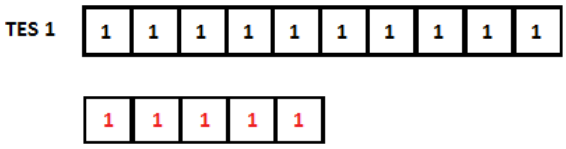


Gambar 1. Kasus terbaik algoritma *Brute Force* saat *pattern* tidak ditemukan

Total perbandingan yang terjadi : m*(n-m+1)

Kompleksitas waktu : O(mn)

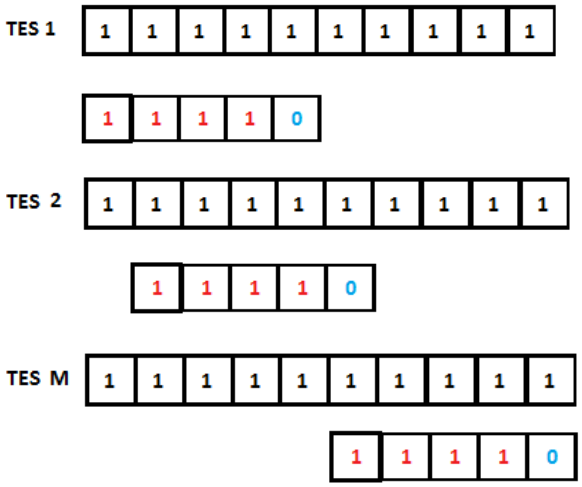
2. Kasus terbaik saat *pattern* ditemukan, maka :



Gambar 2. Kasus terbaik algoritma *Brute Force* saat *pattern* ditemukan

Total perbandingan yang terjadi : m
Kompleksitas waktu : O(m)

3. Bila kasus terburuk terjadi, maka :



Gambar 3. Kasus terburuk algoritma *Brute Force*

Total perbandingan yang terjadi : m*(n-m+1)
Kompleksitas waktu : O(mn)

B. Algoritma Boyer-Moore

Algoritma *Boyer-Moore* adalah algoritma pencocokan untuk *String Matching* dimana pencocokan dimulai tidak dari bagian sebelah kiri *pattern* melainkan dimulai dari bagian sebelah kanan *pattern*. Hal ini dilakukan dengan tujuan mendapatkan informasi yang lebih banyak mengenai teks yang akan dicocokkan dengan *pattern* yang ada.

```
procedure preBmBc(String P, int n, array[0..n-1] of
integer bmBc)
{
  int i;
  for (i := 0 to ASIZE-1){
    bmBc[i] <-- m;
  }
  for (i := 0 to m - 2){
    bmBc[P[i]] <-- m - i - 1;
  }
}
procedure preSuffixes( String P, int n, array[0..n-1] of
integer suff)
{
  int f, g, i;
```

```
    suff[n - 1] <-- n;
    g <-- n - 1;
    for (i <-- n - 2 downto 0)
    {
      if (i > g and (suff[i + n - 1 - f] < i - g))
        suff[i] <-- suff[i + n - 1 - f];
      else
        if (i < g)
          g <-- i;
        f <-- i;
        while (g >= 0 and P[g] = P[g + n - 1 - f])
          --g;
        suff[i] <-- f - g;
      endif
    }
  }
```

```
  procedure preBmGs(String P, int n, array[0..n-1] of
integer bmBc)
  {
    int i,j;
    array [0..RuangAlpabet] of integer suff;

    preSuffixes(x, n, suff);

    for (i <-- 0 to m-1)
      bmGs[i] <-- n
    j <-- 0
    for (i <-- n - 1 downto 0){
      if (suff[i] = i + 1){
        for (j<--j to n - 2 - i){
          if (bmGs[j] = n)
            bmGs[j] <-- n - 1 - i
          }
      }
    }
    for (i <-- 0 to n - 2)
      bmGs[n - 1 - suff[i]] <-- n - 1 - i;
```

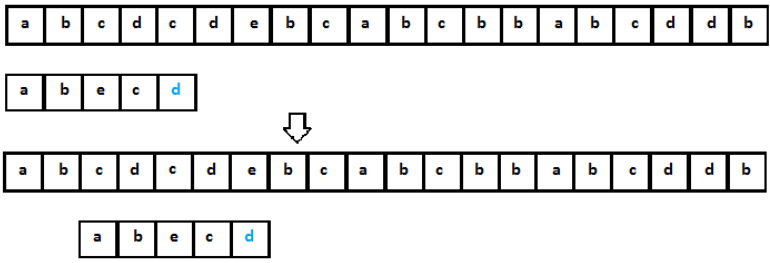
Pseudocode 2.Pra-Pencarian pada algoritma *Boyer-Moore*

```
  procedure BoyerMooreSearch(int m, int n, String P,
String T, int n, array[0..n-1] of boolean ketemu){
          int i, j, shift, bmBcShift, bmGsShift;
          array[0..255] of interger BmBc ;
          array[0..n-1] of interger BmGs ;
          preBmBc(n, P, BmBc);
          preBmGs(n, P, BmGs);
          i<--0;
          while (i<= m-n){
            j:=n-1
            while (j >=0 n and T[i+j] = P[j])
              j<--j-1
            if(j < 0)
              ketemu[i]:=true;
            bmBcShift<-- BmBc[chartoint(T[i+j])]-n+j+1
            bmGsShift<-- BmGs[j]
            shift<-- max(bmBcShift, bmGsShift)
            i<-- i+shift
          }
  }
```

Pseudocode 3. Pencarian pada algoritma *Boyer-Moore*

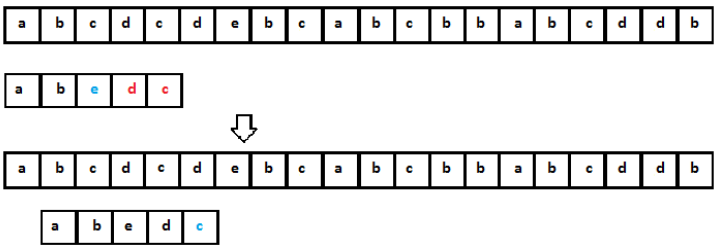
Ada 3 kasus khusus yang dapat terjadi saat melakukan pencarian dengan algoritma *Boyer-Moore* pada suatu *pattern* P dengan teks T, yaitu :

1. Saat P berisi x (karakter yang sedang dicek pada teks), kemudian lakukan lompatan pada P untuk menyamakan *last occurrence* x pada P dengan T[i] (tempat pengecekan sekarang).



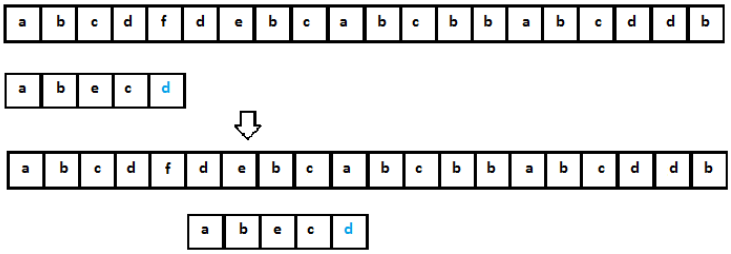
Gambar 4. Kasus saat P berisi x dari T

2. Saat P berisi x (karakter yang sedang dicek pada teks), tetapi lompatan pada P untuk menyamakan *last occurrence* x pada P dengan T[i] (tempat pengecekan sekarang) tidak dapat dilakukan maka lompatan dilakukan sebanyak 1 kali.



Gambar 5. Kasus saat P berisi x dari T dan *last occurrence* dari x pada P tidak dapat di-*align*

3. Bila kejadian yang terjadi tidak termasuk kasus 1 dan 2, maka lakukan lompatan hingga P[1] berada pada T[i+1].



Gambar 6. Kasus saat P tidak berisi x dari T

Algoritma *Boyer-Moore* berjalan pada kompleksitas waktu O(mn + A) dalam kasus terburuk, dengan A adalah domain dari karakter yang ada pada teks dan *pattern* yang akan dibandingkan.

C. Algoritma Knuth-Morris-Pratt

Algoritma *Knuth-Morris-Pratt* adalah sejenis versi *upgrade* dari algoritma *brute force*. Dengan menggunakan algoritma ini, kita dapat melihat bahwa pada algoritma *brute force* terdapat pengulangan langkah yang diambil sehingga terjadi redundansi. Optimasi yang dilakukan oleh algoritma *knuth-morris-pratt* ini adalah dengan cara melangkahi langkah yang seharusnya tidak perlu dilakukan,

```
procedure preKMP(String P,int n,array[0..n] of integer
kmpNext)
{
  int i,j;
 i <-- 0;
 kmpNext[0] <-- -1;
 j <-- kmpNext[0];

 while (i < n) {
   while (j > -1 and not(P[i] = P[j]))
     j <-- kmpNext[j];
   i<-- i+1;
   j<-- j+1;
   if (P[i] = P[j])
     kmpNext[i] <-- kmpNext[j];
   else
     kmpNext[i] <-- j;
  }
}
```

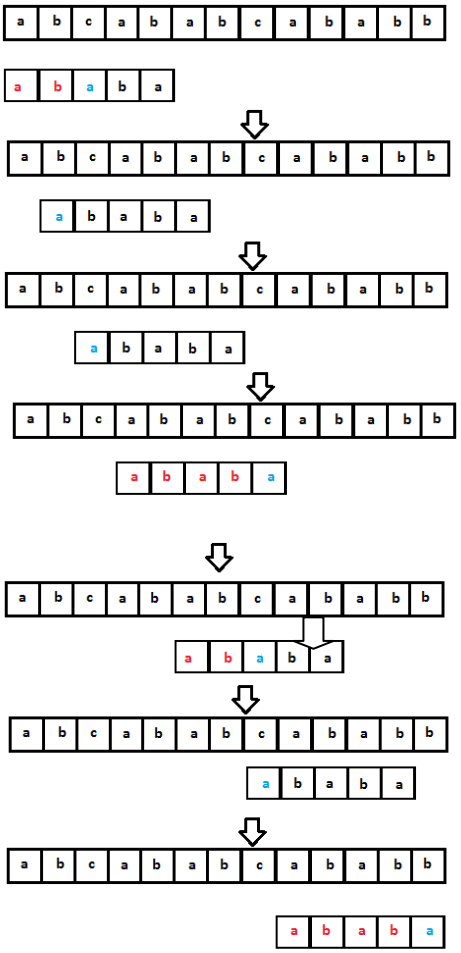
Pseudocode 4.Pra-pencarian pada algoritma *Knuth-Morris-Pratt*

```
procedure KMPSearch(int m, int n, String P, String T,
array[0..m-1] of boolean ketemu)
{
int i, j,next;
array[0..n] of interger kmpNext;

preKMP(n, P, kmpNext) ;
i<--0
while (i<= m-n) {
   j<--0
   while (j < n and T[i+j] = P[j]) {
     j<--j+1
   if(j >= n)
     ketemu[i]:=true;

   next<-- j - kmpNext[j]
   i<-- i+next
   }
}
```

Pseudocode 5. Pencarian Algoritma *Knuth-Morris-Pratt*



Gambar 7. Contoh pencarian pada algoritma *Knuth-Morris-Pratt*

Algoritma *Knuth-Morris-Pratt* memiliki kompleksitas waktu O(m+n+A) pada umumnya dan akan meningkat bila alphabetnya juga meningkat

III. PEMBAHASAN

Pada mesin pencari, pencarian *query* dari pengguna tidak seutuhnya dilakukan hanya dengan melakukan *string matching* dari *query* terhadap data yang ada pada basis data mesin pencari. Pembahasan kali ini akan fokus hanya pada bagian pencocokan kata yang ada pada *query*.

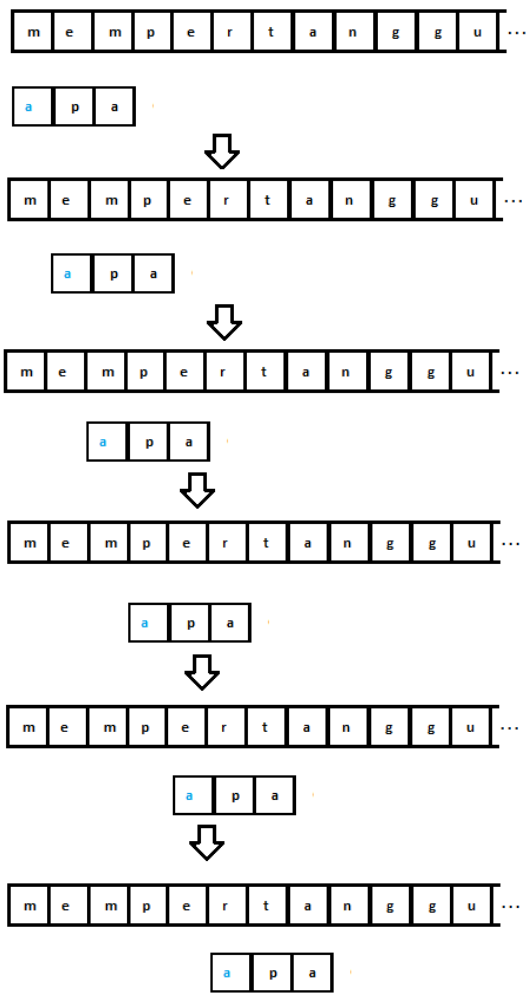
Query dari pengguna biasanya merupakan Alfabet yang sangat heterogen, untuk pembahasan kali ini *query* yang akan dipakai untuk pencarian adalah "apa". *Query* yang ada tersebut akan dicocokkan dengan begitu banyak kata / frasa yang telah mempunyai indeks terhadap website yang mengandung kata / frasa tersebut. Misalkan kita mempunyai basis data kata, seperti tabel di bawah ini:

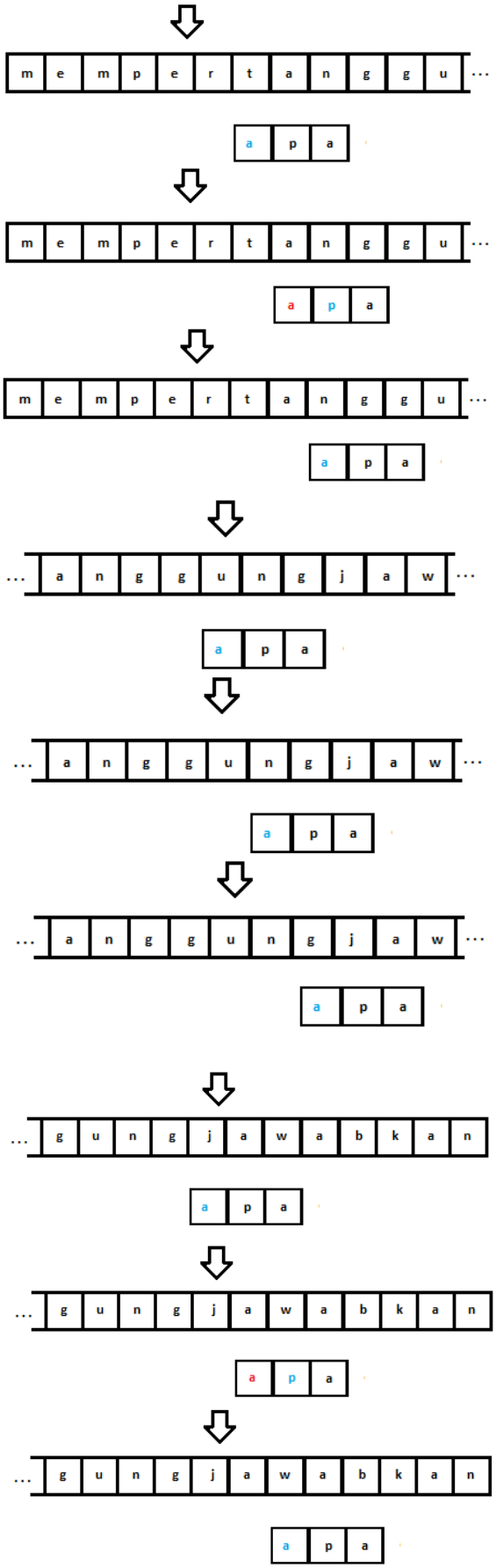
No	Kata	Indeks
1	Matematika	1,4,7,2
2	Kapak	3,6
3	Kalkulus	5,6
4	Algoritma	3,5
5	Mempertanggung jawabkan	2,8
6	Bapak	1,5
7	Strategi	1,4
8	Kapal	2
9	Melakukan	4,7

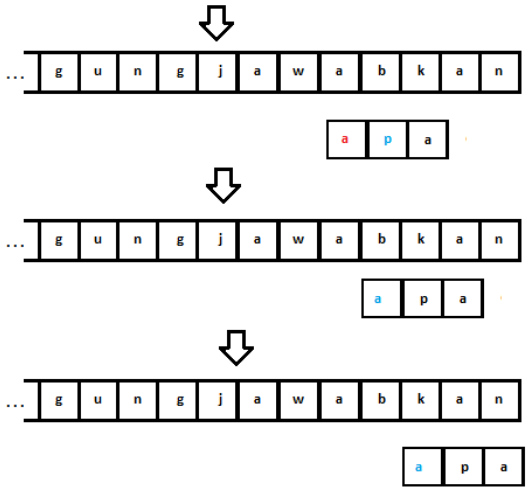
Untuk tiap kata yang ada pada tabel tersebut akan dilakukan pencocokan dengan terhadap *query* "apa" dengan algoritma yang ditentukan.

Berikut adalah algoritma yang digunakan dan jumlah perbandingan yang akan dilakukannya beserta cara perbandingannya terhadap kata "Mempertanggungjawabkan":

1. Algoritma *Brute Force*



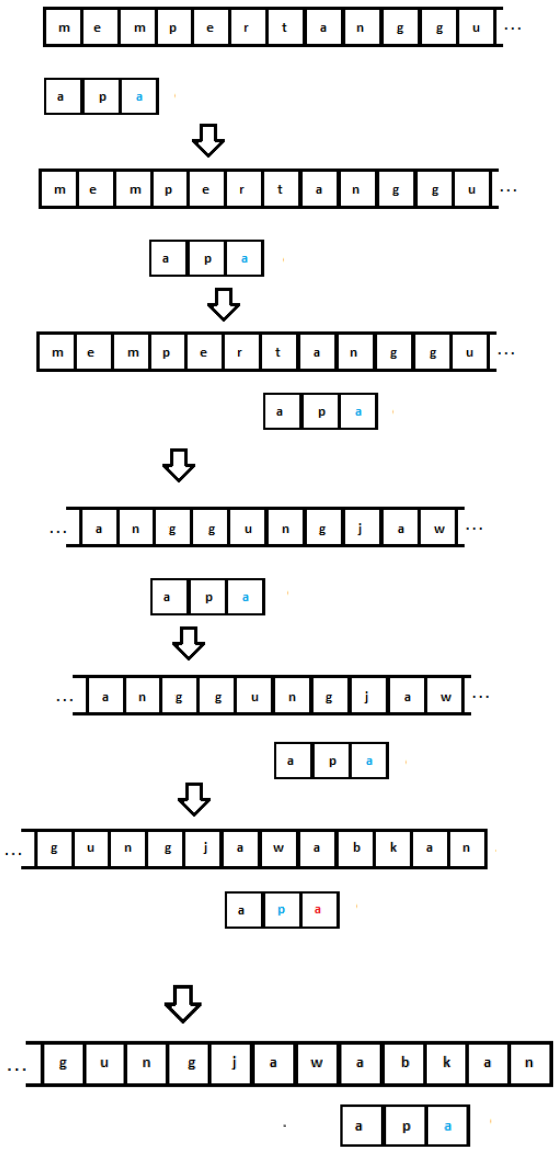




Gambar 8. Pengecekan *pattern* "apa" pada teks "mempertanggungjawabkan" dengan *Brute Force*

Jumlah perbandingan yang dilakukan untuk semua isi tabel sebanyak 104 kali

2. Algoritma *Boyer-Moore*



Gambar 9. Pengecekan *pattern* "apa" pada teks "mempertanggungjawabkan" dengan *Boyer-Moore*

Jumlah perbandingan yang dilakukan untuk semua isi tabel sebanyak 36 kali

3. Algoritma *Knuth-Morris-Pratt*
Perbandingan yang dilakukan untuk algoritma ini pada percobaan kali ini sama dengan perbandingan dengan algoritma *brute force*.
Jumlah perbandingan yang dilakukan untuk semua isi tabel sebanyak 104 kali

IV. KESIMPULAN

Pencocokan kata pada mesin pencari paling optimal dengan menggunakan *Boyer-Moore*. Hal ini disebabkan algoritma ini menggunakan informasi yang didapatkan selama langkah pra-persiapan untuk melangkahi bagian-bagian dari teks yang telah "dianggap" tidak memenuhi *pattern* yang dimasukkan.

Hasil perhitungan dengan menggunakan algoritma *Brute Force* sama dengan hasil perhitungan dengan menggunakan algoritma *Knuth-Morris-Pratt* disebabkan karena *pattern* yang dimasukkan memiliki panjang yang sangat pendek sehingga menyebabkan tidak dapatnya dilakukan pelompatan langkah untuk algoritma *Knuth-Morris-Pratt.*

DAFTAR PUSTAKA

[1] Rinaldi Munir, Diktat Strategi Algoritma, 2009.
[2] http://en.wikipedia.org/wiki/String_matching diakses pada 18 Mei 2014
[3] http://en.wikipedia.org/wiki/Web_search_engine diakses pada 18 Mei 2014
[4] http://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string_search_algorithm diakses pada 18 Mei 2014
[5] http://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm diakses pada 18 Mei 2014
[6] http://www.inf.fh-flensburg.de/lang/algorithmen/pattern/kmpen.htm diakses pada 18 Mei 2014
[7] http://www.cs.utexas.edu/~moore/publications/fstrpos.pdf diakses pada 18 Mei 2014
[8] http://www.cis.uoguelph.ca/~xli/courses/cis2520/c16.pdf diakses pada 18 Mei 2014

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, 17 Mei 2014

Eldwin Christian / 13512002