TUGAS INDIVIDU

STRUKTUR DATA DAN ALGORITMA

Algoritma String Matching

Disusun Oleh:

Mutiara Auliya Khadija

M0513034

JURUSAN INFORMATIKA

UNIVERSITAS SEBELAS MARET

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

SURAKARTA

BAB I

PENDAHULUAN

A. Latar Belakang Masalah

Dewasa ini kemajuan yang sangat pesat di bidang teknologi, terutama dibidang teknologi informasi. Masyarakat banyak menggunakan berbagai gadget untuk mempermudah aktivitasnya. Sehingga mendorong masyarakat untuk mengeluarkan ide- ide kreatif mereka. Maka semakin banyak pula inovasi baru dalam penyajian informasi untuk memenuhi kebutuhan informasi. Begitu pula dalam hal pencarian kata- atau kalimat. Dapat dilihat sekarang ini kemudahan informasi semakin terbuka. Apabila masyarakat tidak mengerti akan suatu hal maka langsung buka website google.com kemudian langsung browsing. Selain itu, untuk mempermudah pencarian kata atau file, user tinggal menggunakan fungsi find atau search. Proses pencarian tersebut salah satunya menggunakan algoritma string matching. Algoritma ini memungkin seseorang untuk mencari suatu kata berdasarkan patter tersendiri. Sehingga, penulis bertujuan untuk membuat program pencarian kata dengan menggunakan Algoritma String Matching.

B. Perumusan Masalah

Adapun perumusan masalahnya yaitu, penulis akan membuat program untuk pencocokan string (String Matching) dengan menggunakan 3 algoritma yaitu Brute Force, KMP (Knuth-Morris-Pratt) dan Byore-Moore.

C. Tujuan Pembuatan Program

Adapun tujuan pembuatan program ini yaitu:

1. Agar masyarakat lebih mudah dalam pencarian kata dalam suatu file menggunakan

Algoritma String Matching

2. Agar masyarakat dapat membandingkan running time dari setiap jenis Algortma String Matching sehingga dapat terpilih mana yang terbaik

3. Untuk memenuhi tugas individu dari mata kuliah Struktur Data Algoritma 2

BAB II

DASAR TEORI

String matching merupakan pencarian sebuah pattern pada sebuah teks. Algoritma string matching adalah algoritma yang ditujukan untuk melakukan string matching. Prinsip kerja algoritma string matching adalah sebagai berikut:

1. Men-*scan teks* dengan bantuan sebuah *window* yang ukurannya sama dengan panjang *pattern.*
2. Menempatkan *window* pada awal *teks.*
3. Membandingkan karakter pada *window* dengan karakter dari *pattern.* Setelah pencocokan (baik hasilnya cocok atau tidak cocok), dilakukan *shft* ke kanan pada *window.* Prosedur ini dilakukan berulang-ulang sampai *window* berada pada akhir teks. Mekanisme ini disebut mekanisme *sliding-window* [l].

Algoritma-algoritma *string* matching mempunyai tiga komponen, yaitu: *pattern,* teks dan alfabet dengan asumsi sebagai berikut:

1. *Pattern,* yaitu deretan karakter yang dicocokkan dengan teks, dinyatakan dengan x[0..m-1], panjang *pattern* dinyatakan dengan m.
2. Teks, yaitu tempat pencocokan *pattern* dilakukan, dinyatakan dengan y[0..n-1], panjang teks dinyatakan dengan n.

Alfabet, yang berisi semua simbol yang digunakan oleh bahasa pada teks dan *pattern*, dinyatakan dengan ∑ dengan ukuran dinyatakan dengan ASIZE. Beberapa jenis algoritma String Matching yaitu Brute Force, KMP (Knuth-Morris-Pratt) dan Byore-Moore.

A. Algoritma *Boyer Moore*

Algoritma Boyer Moore termasuk algoritma *string matching* yang paling efisien dibandingkan algoritma-algoritma *string matching* lainnya.

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

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

1. Menjalankan prosedur preBmBc dan preBmGs untuk mendapatkan inisialisasi.
    a. 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.
    b. 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.
    c. 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. Contoh pada Gambar 2.1, yaitu berapa langkah yang dibutuhkan dari *au(u* = segmen, *a* = karakter di sebelah kiri *u*) ke *cu* yang mempunyai segmen *u* pada *pattern* dengan karakter di sebelah kiri segmen yaitu *c* berbeda dari *a*

dan terletak lebih kiri dari *au.* Hasil dari prosedur preBmGs disimpan pada tabel BmGs.

2. Dilakukan proses pencarian *string* dengan menggunakan hasil dari prosedur preBmBc dan preBmGs yaitu tabel BmBc dan BmGs.

Berikut ini diberikan contoh untuk menjelaskan proses inisialisasi dari algoritma *Boyer Moore* dengan *pattern* **gcagagag** yang akan dicari pada *string* **gcatcgcagagagtatacagtacg**.

1. Dengan prosedur preBmBc, didapatkan jumlah pergeseran pada *pattern* yang dibutuhkan untuk mencapai karakter a,c,g,t dari posisi terkanan. Berdasarkan contoh diketahui untuk mencapai masing-masing karakter tadi dibutuhkan pergeseran sebanyak 1, 6, 2 dan 8.
2. Dengan prosedur preBmGs, dijalankan prosedur *suffix* terlebih dulu. Dengan prosedur *suffix* akan diketahui:

   suff[0] = 1, 1 karakter g posisi 7 cocok dengan 1 karakter g posisi 0.

   suff[1] = 0, karakter g posisi 7 tidak cocok dengan karakter c posisi 1.

   suff[2] = 0, karakter g posisi 7 tidak cocok dengan karakter a posisi 2.

   suff[3] = 2, 2 karakter dimulai dari karakter g posisi 7 cocok dengan 2 karakter dimulai dari karakter g posisi 3, yang artinya karakter a,g posisi 6,7 cocok dengan karakter a,g posisi 2,3.

   suff[4] = 0, karakter g posisi 7 tidak cocok dengan karakter a posisi 4.

   suff[5] = 4, 4 karakter dimulai dari karakter g posisi 7 cocok dengan 4 karakter dimulai dari karakter 5, artinya karakter a,g,a,g posisi 4,5,6,7 cocok dengan karakter a,g,a,g posisi 2,3,4,5.

   suff[6] = 0,karakter g posisi 7 tidak cocok dengan karakter a posisi 6.

   suff[7] = 8, 8 karakter g,c,a,g,a,g,a,g posisi 0,1,2,3,4,5,6,7 cocok dengan 8 karakter g,c,a,g,a,g,a,g posisi 0,1,2,3,4,5,6,7.

3. Dengan prosedur BmGs akan didapatkan:

   0 1 2 3 4 5 6 7

   g c a g a g a g

   bmGs[0]= 7, karakter ke-0 g adalah karakter sebelah kiri segmen cagagag.Tidak ada segmen cagagag lain dengan karakter sebelah kiri bukan g maka digeser 7 langkah.

   bmGs[1]= 7, karakter ke-1 c adalah karakter sebelah kiri segmen agagag. Tidak ada segmen agagag lain dengan karakter sebelah kiri bukan c maka digeser 7 langkah.

   bmGs[2]= 7, karakter ke-2 a adalah karakter sebelah kiri segmen gagag. Tidak ada segmen gagag lain dengan karakter sebelah kiri bukan a maka digeser 7 langkah.

   bmGs[3]= 2. karakter ke-3 g adalah karakter sebelah kiri segmen agag. Karena ada segmen agag posisi 2,3,4,5 dengan karakter sebelah kiri bukan g yaitu c posisi 1 maka digeser 2 langkah.

   bmGs[4]= 7, karakter ke-4 a adalah karakter sebelah kiri segmen gag. Karena tidak ada seamen gag lain dengan karakter sebelah kiri bukan a maka digeser 7 langkah.

   bmGs[5]= 4. karakter ke-5 g adalah karakter sebelah kiri seamen ag. Karena ada segmen ag posisi 2,3 dengan karakter sebelah kiri bukan g yaitu c posisi 1 maka digeser 4 langkah.

   bmGs[6]= 7, karakter ke-6 a adalah karakter sebelah kiri segmen yaitu a posisi 7. Karena tidak ada segmen g dengan karakter sebelah kirinya bukan a maka digeser 7 langkah.

   bmGs[7]= 1, karakter ke-7 g adalah karakter sebelah kiri segmen dan karena segmen tidak ada maka digeser 1 langkah

B. Algoritma Brute Force

Algoritma brute force string match merupakan algoritma yang paling sederhana untuk memecahkan masalah string match. Cara kerja algoritma ini adalah dengan

mencoba setiap posisi pattern(kata yang akan dicocokkan) terhadap teks, kemudian dilakukan proses pencocokan setiap katakter dan teks pada posisi tersebut dari kiri ke kanan. Misalnya I menyatakan suatu indeks pada teks dan j merupakan indeks dari pola anatara 0 sampai m-1. Windows ditaruh di posisi paling kiri teks lalu dilakukan perbandingan pertama antara T[0] dan P[0]. Apabila terjadi kecocokan maka masing-masing indeks akan dinaikkan satu satu. Jika tidak terjadi kecocokan maka indeks j akan dikembalikan ke awal pola yaitu j=0 dan windows atau pattern akan digeser satu ke kanan sehingga indeks I sama dengan satu. Jika terjadi ketidakcocokan maka patternnya pasti akan digeser ke kanan sebanyak satu.

C. Algoritma Knuth-Morris-Pratt

Algoritma ini juga dilakukan suatu perbandingan karakter di teks dan karakter pada pola kiri ke kanan. Hampir sama dengan algoritma Brute Force, apabila algoritma Brute Force pada pergeserannya. Tetapi algoritma Knuth Morris Pratt ini bagaimana memanfaatkan karakter pola yang sudah diketahui di dalam teks sampai terjadinya ketidakcocokan untuk melakukan pergeseran. Misalk string tesk T memiliki panjang n dengan indeks I lalu string P memiliki panjang m ddengan indeks j. Jika terjadi ketidakcocokan maka untuk menentukan besar pergeseran dengan mencari prefix terpanjang dari pola P[0...j-1].

BAB III

ANALISIS DAN RANCANGAN APLIKASI

Berdasarkan landasan teori maka penulis dapat mengetahui sifat, ciri- ciri serta cara kerja dari masing masing algoritma String Matching yaitu Brute Force, KMP (Knuth-Morris-Pratt) dan Byore-Moore. Program atau aplikasi yang akan penulis buat harus memenuhi ketentuan- ketentuan sebagai berikut:

a. Input berupa T (Teks) dari file eksternal misal file dengan format .txt dll, selain itu juga input P (Pattern) yakni string yang akan dicari.

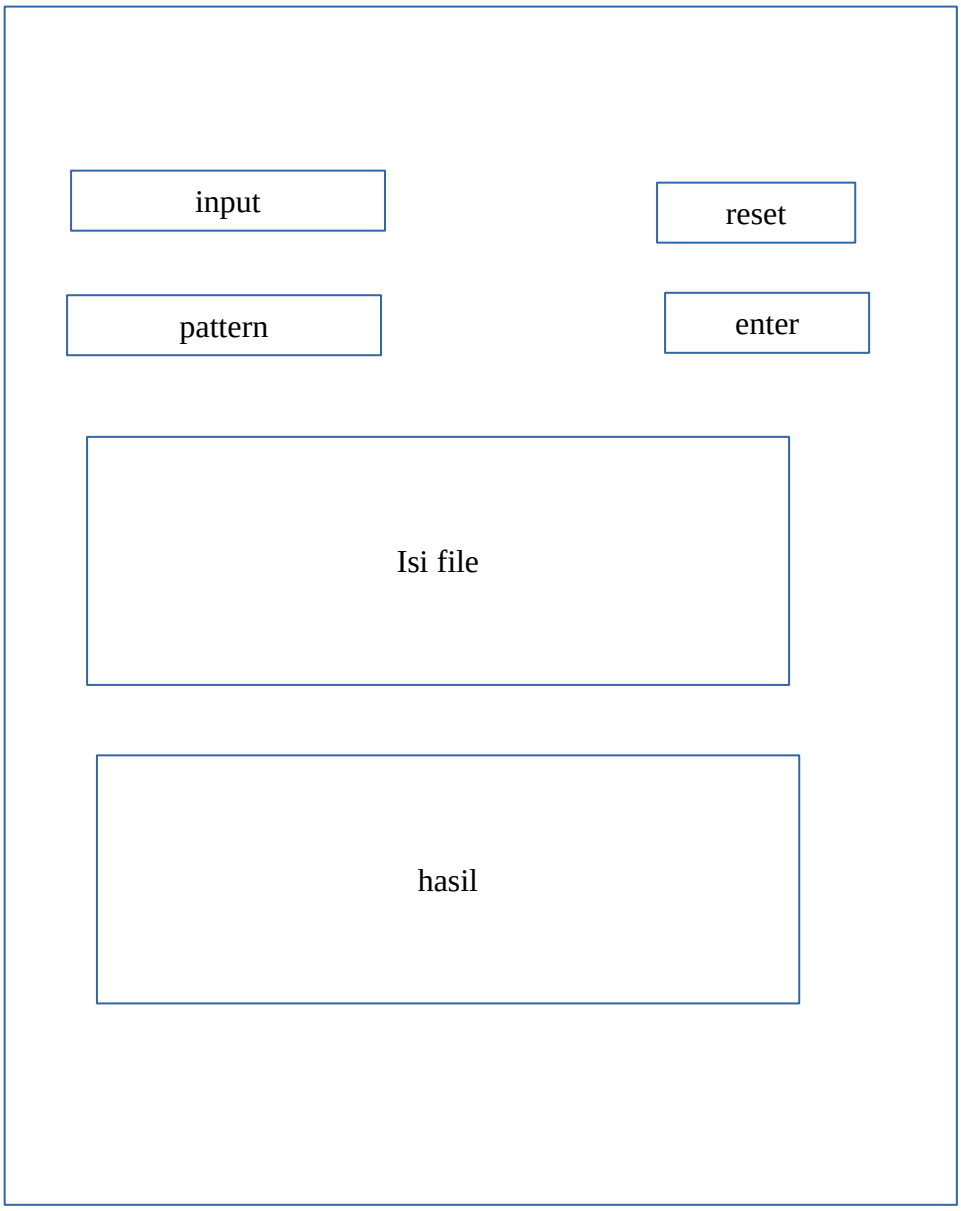
b. Output dari aplikasi adalah hasil status dari pencocokan string apakah berhasil atau tidak, jika berhasil program akan menampilkan posisi/lokasi awal dari P (pattern) di T

c. Aplikasi juga harus menampilkan hasil running time dari ketiga algoritma tersebut.

d. Aplikasi menyediakan pilihan untuk mengulangi input T (Teks) maupun P(Pattern) yang berbeda.

c. Aplikasi yang dibuat harus berbasis grafis (GUI)

Berdasarkan ketentuan berikut maka penulis harus membuat program menggunakan bahasa pemrograman java berbasis GUI. Maka penulis pertama membuat perancangan desain GUI. Desain yang penulis buat memiliki berbagai komponen diantaranya input, enter, dan reset yang merupakan Jbutton. Kemudian terdapat Pattern yang merupakan JtextField. Dan terdapat Isi File serta Hasil yang merupakan Jtextarea. Maka, desain interface dari program dapat dilihat seperti gambar berikut.



Penulis menggunakan JFIlechooser sebagai inputannya. Jadi nanti user tinggal memilih file yang diinginkan berbasis txt atau doc. Tidak usah menuliskan alamat direktorinya. Jadi, pertama user akan memilih file apa yang akan dieksekusi. Kemudian user memasukkan pattern nya. Lalu di cek dalam class engine. Class ini terpisah dari class gui. Di dalam class engine terjadi pengecekan inputan, pembacaan data dari file. Kemudian data dari file tersebut akan diproses oleh method algoritma bruteforce, KMP dan Bayer Moore yang telah didefinisikan. Isi file akan dicocokan dengan pattern. Apabila pattern tersebut tersedia dalam isi file atau data maka isi file akan dibaca dan terlihat dalam output

isi file. Setelah itu hasil dari pengecekan akan dimasukkan ke button enter. Berdasarkan hasil tersebut dapat dilakukan penghitungan running time untuk menentukan algoritma mana yang paling efisien. Setelah itu program akan mencetak hasil akhirnya pada JtextArea Hasil. Di dalam desin juga terdapat reset yaitu dengan mengkosongkan semua isi pada field apabila user ingin melakukan pencarian data lagi.

BAB IV

HASIL DAN PEMBAHASAN

Berdasarkan analisa dan percancangan program di atas maka penulis membuat program java dengan 3 class. Yaitu String_matching, engine2, dan string_GUI. Class String_matching digunakan untuk class main atau class utama agar program dapat dijalankan. Adapun penampakannya adalah sebagai berikut:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
package string_matching;

/**
 *
 * @author root
 */
public class String_matching {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        stringGUI start = new stringGUI();
        start.main(args);
    }
}
```

Kemudian masuk ke class engine1. Di dalam class engine1 ini merupakan engine dari GUI yang telah didefinisikan.

```
 * and open the template in the editor.
 */
package string_matching;

/**
 *
 * @author root
 */
import java.io.BufferedInputStream;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.InputStream;
import java.io.InputStreamReader;
```

Di dalam class engine1 terdapat method fileInput yang berfungsi untuk mensinkronkan antara filechooser di dalam GUI dengan di dalam engine sehingga nantinya akan bisa di proses oleh method- method lainnya. Di sini penulis menggunakan bufferedreader untuk membaca isi file yang telah dipilih. Program akan membaca isi file tersebut sampai hasbi atau null dan menyimpannya dalam obyek sbuffer. Nanti di dalam string_GUI terjadi insialisasi sbuffer menjadi variabel text.

```
public class engine1 {
    String fileInput(String path) {
        InputStream is= null;
        InputStreamReader isr = null;
        BufferedReader br = null;
        File file = new File(path);
        StringBuffer sbuffer = new StringBuffer("");
        try {
            is = new FileInputStream(file);
            isr = new InputStreamReader(is);
             br = new BufferedReader(isr);

                 while(is.available() != 0){
                 sbuffer.append(br.readLine());
                 }
                 is.close(); isr.close(); br.close();
                }
                catch (FileNotFoundException e) {
                }
                catch (IOException e){
            }
        String str = sbuffer.toString();
        return str;
        }
```

Kemudian terdapat method bmMatch yang merupakan algoritma Bayer Moore. Di dalam algoritma ini terjadi pengecekan text dan pattern sesuai dengan algortima Bayer Moore.

```
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;
    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;
}
```

Setelah terjadi pengecekan dengan algoritma Bayer Moore maka text akan diproses lagi menggunakan algoritma Brute Force. Sama seperti algoritma Bayer Moore tadi pattern yang diinputkan user akan dicocokkan dengan text. Proses pencocokan dari kiri ke kanan. Apabila terjadi kecocokan maka masing- masing indeks akan dinaikkan satu satu.

```
int brute (String text, String pattern) {
    int n = text.length();
    int m = pattern.length();
    int j;
    for(int i = 0; i <= (n-m); i++) {
        j = 0;
        while((j<m)&&(text.charAt(i+j)==pattern.charAt(j)))
            j++;
        if(j == m)
            return i;
    }
    return -1;
}
```

Setelah melalui algoritma Brute Force maka akan diproses lagi ke Algoritma KMP. Knuth-Morris-Pratt.

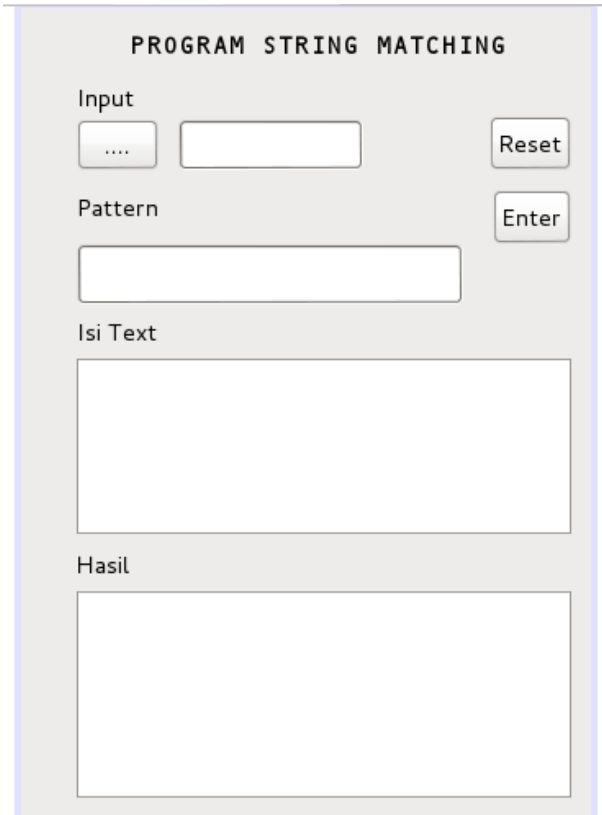
```
static int kmpMatch(String text, String pattern) {
    int n = text.length();
    int m = pattern.length();
    int fail[] = computeFail(pattern);
    int i = 0;
    int j = 0;
    while(i < n) {
        if(pattern.charAt(j) == text.charAt(i)) {
            if(j == m -1)
                return i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

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

`Dalam Algoritma Knuth-Morris-Pratt (KMP), untuk setiap karakter yang dibandingkan kita bisa memutuskan apakah berhasil atau gagal.

```
    static int[] buildLast(String pattern) {
        int last[] = new int[128];
        for(int i = 0; i < 128; i++)
            last[i] = -1;
        for(int i = 0; i < pattern.length(); i++)
            last[pattern.charAt(i)] = 1;
        return last;
    }

}
```

Setelah di eksekusi dalam engine1 maka penulis juga membuat class string_GUI untuk membuat desain interface guinya. Di dalam gui pertama, buat desain guinya terlebih dahulu.



Seperti pada perancangan sebelumnya bahwa penulis akan membuat berbagai

komponen diantaranya input, enter, dan reset yang merupakan Jbutton. Kemudian terdapat Pattern yang merupakan JtextField. Dan terdapat Isi File serta Hasil yang merupakan Jtextarea. Pertama kita mengindentifikasikan obyek engine1 tadi ke dalam GUI supaya engine1 dapat sinkron dengan string_GUI. Dengan cara inisialisasi engine1 dengan obyek mesin.

```
 */
package string_matching;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.JFileChooser;
/**
 *
 * @author root
 */
public class stringGUI extends javax.swing.JFrame {
//JFileChooser chooser = new JFileChooser();
   engine1 mesin = new engine1();
/**
     * Creates new form stringGUI
     */
    StringBuffer sbuf = new StringBuffer("");
     public stringGUI() {
         initComponents();
     }


    */
    @SuppressWarnings("unchecked")
    Generated Code


    private void patternActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
    }
```

Lalu pada design GUI masuk ke jButton input. Penulis memasukkan sintak seperti berikut ke dalam action performednya.

```
private void inputActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:

   int jes=jFileChooser1.showOpenDialog(this);
    if(jes == JFileChooser.APPROVE_OPTION)
    {
        String f = jFileChooser1.getSelectedFile().toString();
        //File f= chooser.getSelectedFile();
        //jLabel1.setText(f.getPath());
        jLabel1.setText(f);
        File file = new File(f);
       try {
           konten.read(new FileReader(file.getAbsolutePath()), null);
       } catch (FileNotFoundException ex) {
           Logger.getLogger(stringGUI.class.getName()).log(Level.SEVERE, null, ex);
       } catch (IOException ex) {
           Logger.getLogger(stringGUI.class.getName()).log(Level.SEVERE, null, ex);
       }
```

Berdasarkan syntak tersebut apabila kita mengklik button input maka kursor akan dibawa ke pemilihan input. Jadi pertama buat obyek jFileChooser1 terlebih dahulu. Kemudian program akan menandai file yang anda pilih. Tanda tersebut akan dimasukkan kedalan jLabel1. Kemudian file yang telah dipilih akan di read di dalam engine1 dalam method fileInput. Setelah di read maka akan diproses oleh method- method algoritma Brute Force, Bayer Moore dan Knuth-Morris-Pratt. Setelah itu masuk ke jButton enter.

```
private void enterActionPerformed(java.awt.event.ActionEvent evt) {
    konten.setText(mesin.fileInput(input.getText()));
    long bruteStart = System.nanoTime();
    int pos = mesin.brute(mesin.fileInput(input.getText()), pattern.getText());
    long bruteEnd = System.nanoTime() - bruteStart;
    sbuf.append(pos + "\nUsing Brute Force in : " + bruteEnd + " nanoseconds");
    long kmpStart = System.nanoTime();
    int pos2 = mesin.kmpMatch(mesin.fileInput(input.getText()), pattern.getText());
    long kmpEnd = System.nanoTime() - kmpStart;
    sbuf.append("\nUsing KMP in : " + kmpEnd + " nanoseconds");
    long boyerStart = System.nanoTime();
    int pos3 = mesin.bmMatch(mesin.fileInput(input.getText()), pattern.getText());
    long boyerEnd = System.nanoTime() - boyerStart;
    sbuf.append("\nUsing Boyer-Moore in : " + boyerEnd + " nanoseconds");
    String str = sbuf.toString();
    if(pos3 == 0) {
        hasil.setText("Pattern not found!");
    } else
        hasil.setText(str);
}
```

Apabila jButton enter diklik maka program akan mengeksekusi semua perintah. Pertama pastinya masuk ke obyek engine1 yaitu mesin di dalam fileInput. Kemudian setelah masuk ke engine1 maka algoritma pertama yaitu Brute Force akan dijalankan. Setelah dijalankan maka program akan masuk ke string_GUI lalu dihitung running time

nya menggunakan system.NanoTime. Terdapat variabel pos pertama, lalu masukkan fileInput text yang ada di file masuk ke field input lalu text pattern yang diinputkan masuk ke pattern. Dan semua itu akan dieksekusi oleh engine1. Maka bruteStart akan dikurangi dengan system.NanoTime. Setelah dihitung maka akan muncul bruteEnd yang merupakan hasil running time. Begitu juga dengan algoritma KMP dan algoritma Bayers Moore. Prinsipnya sama. Yaitu pertama insialisasi dulu fileInputnya yaitu berupa text yang ada di dalam input(isi file) dan pattern(pattern yang dimasukkan). Kemudian system.NanoTime dikurangi dengan insialisi tadi. Setelah itu muncul hasilnya dan dimasukkan ke dalam field mana yang ada di string_GUI.

Lalu terdapat button reset yang berfungsi untuk mengembalikan nilai kembali seperti semula.

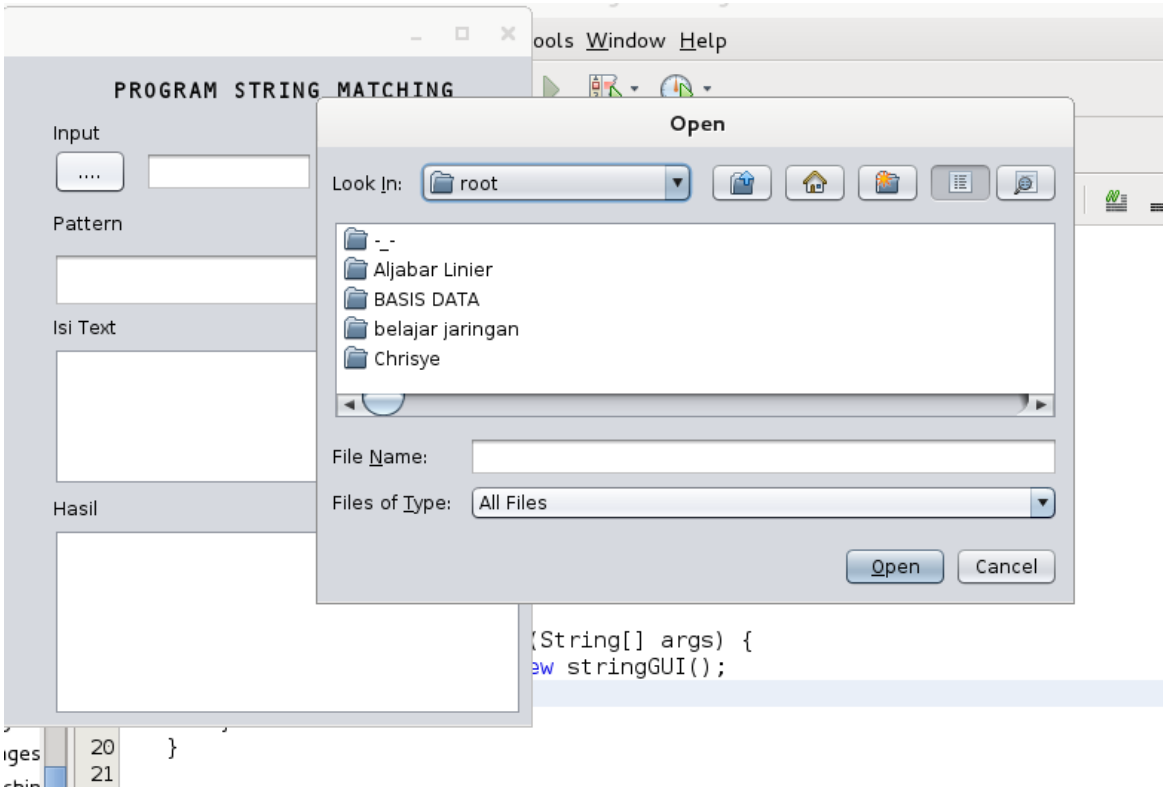
```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
    jButton2.setText("    ");

}

private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}

/**
```

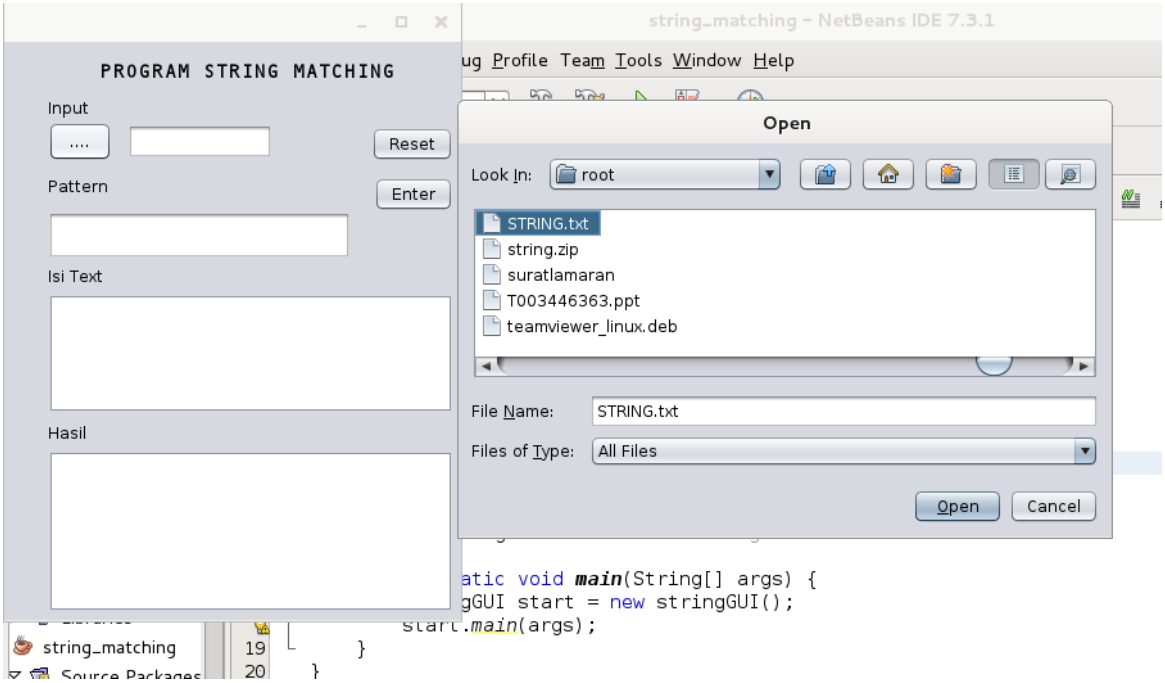
Setelah itu ada fungsi main dan fungsi bawaan dari GUI nya yang merupakan insialisasi dari frame dan form GUI.

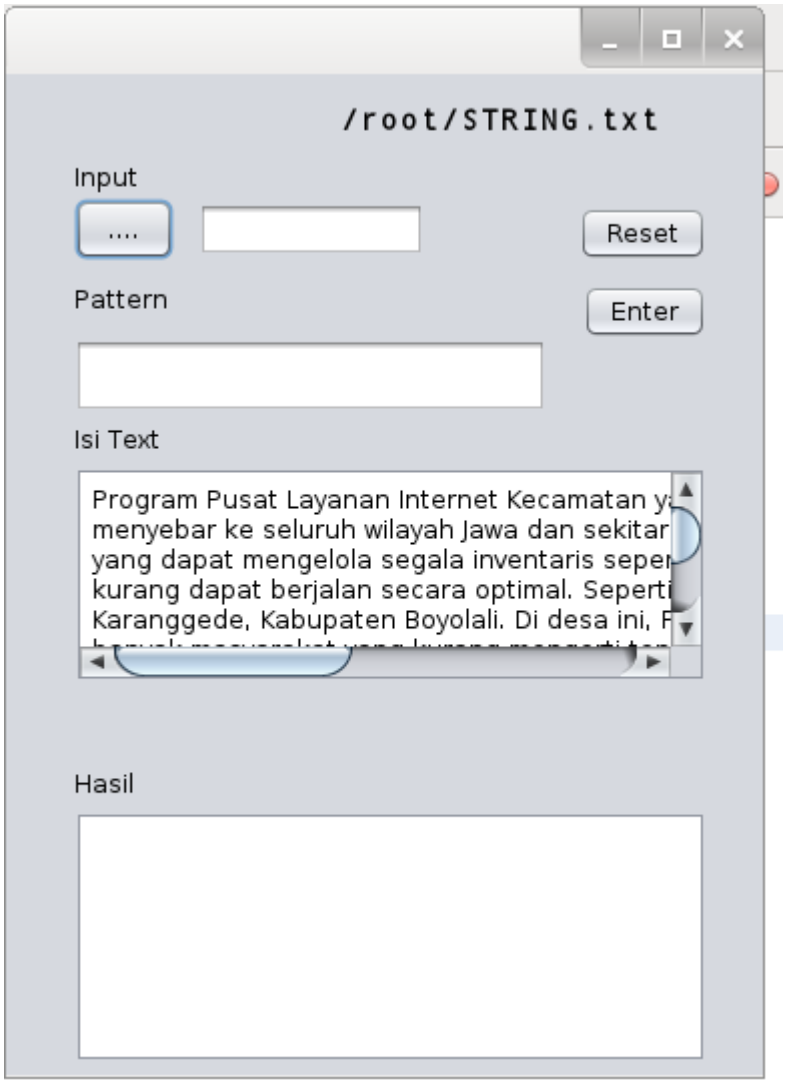
```
 */
public static void main(String args[]) {
    /* Set the Nimbus look and feel */
    Look and feel setting code (optional)

    /* Create and display the form */
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new stringGUI().setVisible(true);
        }
    });
}
```

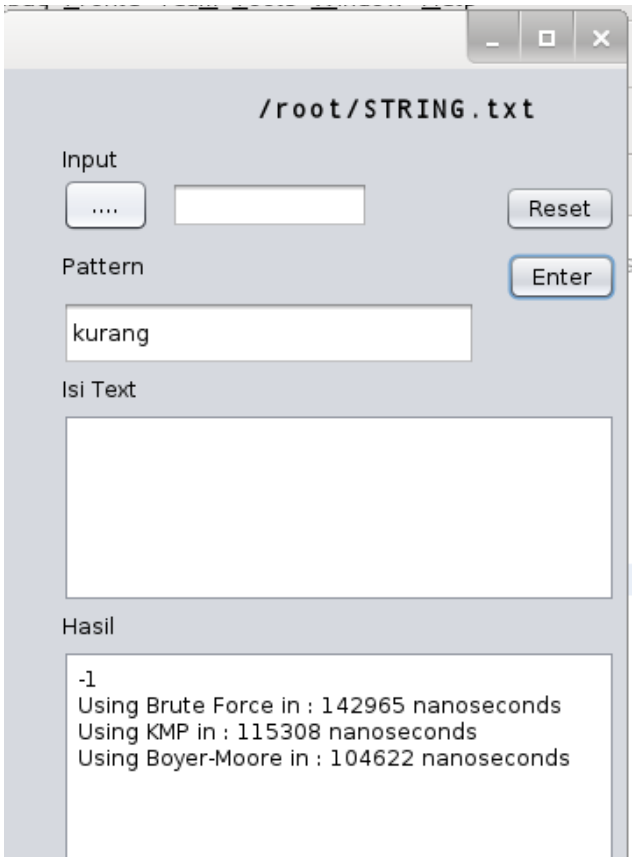
```
    }
    // Variables declaration - do not modify
    private javax.swing.JButton enter;
    private javax.swing.JTextArea hasil;
    private javax.swing.JButton input;
    private javax.swing.JButton jButton2;
    private javax.swing.JFileChooser jFileChooser1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextArea konten;
    private javax.swing.JTextField pattern;
    // End of variables declaration
}
```

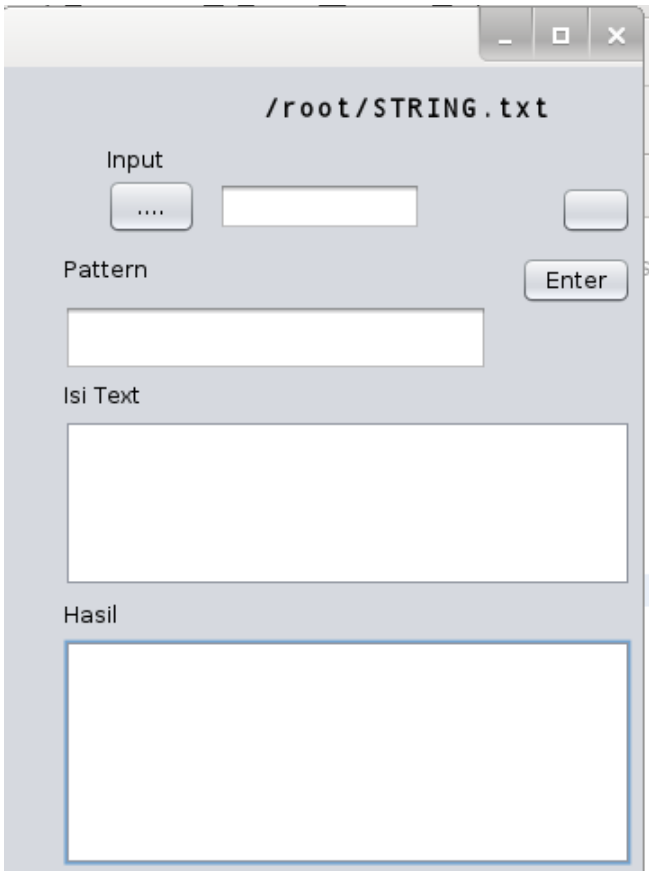
Apabila di run maka hasilnya akan sebagai berikut,




PROGRAM STRING MATCHING
Input
....
Reset
Pattern
Enter
Isi Text
Hasil
string_matching - NetBeans IDE 7.3.1
ug Profile Team Tools Window Help
Open
Look In:
root
STRING.txt
string.zip
suratlamaran
T003446363.ppt
teamviewer_linux.deb
File Name:
STRING.txt
Files of Type:
All Files
Open
Cancel
atic void main(String[] args) {
gGUI start = new stringGUI();
start.main(args);
string_matching
19 }
20 }


/root/STRING.txt
Input
....
Reset
Pattern
Enter
Isi Text
Program Pusat Layanan Internet Kecamatan y
menyebar ke seluruh wilayah Jawa dan sekitar
yang dapat mengelola segala inventaris seper
kurang dapat berjalan secara optimal. Seperti
Karanggede, Kabupaten Boyolali. Di desa ini, P
Hasil


/root/STRING.txt
Input
....
Reset
Pattern
Enter
kurang
Isi Text
Hasil
-1
Using Brute Force in : 142965 nanoseconds
Using KMP in : 115308 nanoseconds
Using Boyer-Moore in : 104622 nanoseconds


/root/STRING.txt
Input
....
Pattern
Enter
Isi Text
Hasil

BAB 5

KESIMPULAN DAN SARAN

A. Kesimpulan

Berdasarkan uraian di atas maka dapat diambil kesimpulan bahwa:

1. Untuk memudahkan suatu pencarian kata maka dapat dibuat suatu Program String Matching berdasarkan algoritma string matching yaitu Brute Force, KMP (Knuth-Morris-Pratt) dan Byore-Moore.

2. Berdasarkan program yang telah dibuat maka diantara ketiga jenis algoritma String Matching tersebut yang memiliki running time paling efisien yaitu algoritma Byore-Moore.

B. Saran

Adapun saran yang diajukan penulis yaitu,

1. Program pencarian kata berdasarkan algoritma string matching ini perlu ditinjau ulang kembali karena dikhawatirkan masih banyak bug- bug

2. Untuk kedepannya program pencarian kata seperti ini perlu dikembangkan untuk membuat suatu aplikasi anti plagiasi

DAFTAR PUSTAKA

Utomo Darmawan, Harjo Eric Wijaya, Handoko.2012.PERBANDINGAN ALGORITMA STRING SEARCHING BRUTE FORCE, KNUTH MORRIS PRATT, BOYER MOORE, DAN KARP RABIN PADA TEKS ALKITAB BAHASA INDONESIA

Hartoyo Eko Gunocipto,Vembrina Yus Gias, Meilana Anggia Ferdina.2011.Analisis Algoritma Pencarian String (String Matching)

http://id.wikipedia.org/wiki/Algoritma_pencarian_string diakses pada 7 Desember 2014 pukul 09.00

http://www.tutorialspoint.com/java/io/bufferedreader_read_char.htm diakses pada 7 Desember 2014 pukul 09.30

http://www.diskusiweb.com/discussion/30036/menampilkan-isi-file-ke-textarea/p1 diakses pada 7 Desember 2014 pukul 09.40