BAB 1

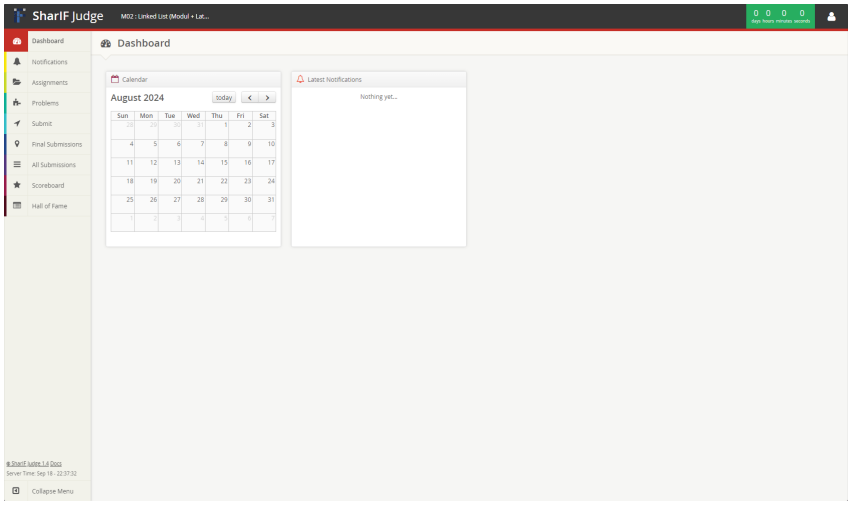
PENDAHULUAN

1.1 Latar Belakang

Ujian merupakan sebuah alat bantu untuk menilai pemahaman pelajar tentang ilmu yang diberikan oleh pengajar. Salah satu ujian yang diberikan kepada pelajar informatika adalah ujian koding yang biasanya dinilai berdasarkan ketepatan algoritma yang dipakai. Tetapi melakukan penilaian untuk setiap kode merupakan sebuah hal yang sulit untuk dilakukan karena dibutuhkannya waktu yang lama. Maka dari itu website *judge* dibuat untuk memudahkan pekerjaan pengajar.

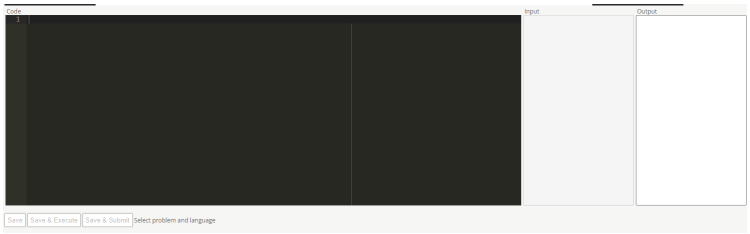
Judge merupakan sebuah website yang akan menilai sebuah kode dengan menjalankannya berdasarkan masukkan yang ditentukan dan menyamakan keluaran dari kode dengan keluaran yang sudah ditetapkan oleh pembuat soal dalam kurun waktu yang ditetapkan. Oleh karena itu, kode yang dibuat harus dapat mencakupi waktu yang diberikan dengan mengunakan algoritma yang tepat. Bukan hanya menilai dengan keluaran yang tetap tetapi *judge* juga dapat mengunakan kode yang sudah dibuat oleh pengajar dan membandingkannya dengan keluaran kode yang di kumpulkan.

Sudah banyak perguruan tinggi informatika yang mengunakan website *judge* dalam pemeriksaan kode pelajar termasuk perguruan UNPAR untuk penilai kode dari para mahasiswanya. Judge yang digunakan adalah SharIF-Judge [1] yang dimodifikasi oleh Stillmen Vallian terhadap Sharif-Judge [2] buatan Mohammad Javad Naderi dengan *framework* CodeIgniter dan Bash. Gambar 1.1 merupakan halaman utama setelah masuk ke dalam website SharIF-Judge.



Gambar 1.1: Tampilan Awal SharIF Judge

Tugas akhir ini merupakan sebuah pengembangan lanjutan dari tugas akhir yang bertopik "Implementasi editor kode pada Sharif Judge" [3] oleh Nicholas Aditya Halim. Tugas akhir tersebut menceritakan bahwa SharIF-Judge tidak memiliki kemampuan untuk mengawasi proses pembuatan kode program karena para mahasiswa menggunakan aplikasi eksternal untuk pembuatan kode program tersebut. Sehingga dibuatnya modifikasi terhadap SharIF-Judge untuk menambahkan



Gambar 1.2: Tampilan editor kode pada SharIF-Judge

Intergrated Development Enviroment (IDE), sebuah aplikasi untuk mengedit, mengompilasi, dan menjalankan kode program pada SharIF-Judge dengan editor kode bernama Ace [4]. Gambar 1.2 merupakan tampilan editor kode yang sudah diimplementasikan pada SharIF-Judge.

Tetapi SharIF Judge masih tidak memiliki kemampuan untuk mengawasi proses pembuatan kode program pada aplikasi eksternal maupun IDE dalam SharIF Judge. Maka dari itu tugas akhir ini menambahkan fitur pada SharIF Judge dengan merekam ketikan pada IDE yang tersedia di SharIF-Judge untuk membantu pengawasan dengan merekam dan memutar ulang ketikan mahasiswa. Tugas akhir ini akan membuat pengawasan terhadap kegiatan kuliah lebih mudah untuk pengawas dan dapat menjadi bukti kecurangan jika dibutuhkan.

1.2 Rumusan Masalah

Rumusan Masalah yang akan dibahas pada tugas akhir ini adalah:

1. Bagaimana mengimplementasikan perekaman dan pemutaran ulang ketikan mahasiswa pada IDE SharIF-Judge?
2. Bagaimana cara menyimpan data pemutaran ulang mahasiswa secara rutin dengan otomatis dan tidak mengambil penyimpanan *database* sangat besar?
3. Bagaimana tanggapan pengguna terhadap implementasi perekaman dan pemutaran ulang kode ketikan pada SharIF Judge?

1.3 Tujuan

Tujuan yang ingin dicapai skripsi ini adalah sebagai berikut:

1. Mengimplementasikan perekaman dan pemutaran ulang ketikan mahasiswa pada IDE SharIF-Judge.
2. Mencari cara penyimpanan data efektif dan mengimplementasikannya pada perekaman dan pemutaran ulang ketikan.
3. Mendapatkan umpan balik dari tanggapan pengguna terhadap perekaman dan pemutaran ulang ketikan mahasiswa pada SharIF-Judge.

1.4 Batasan Masalah

Pada pengerjaan tugas akhir ini terhadap batasan sebagai berikut:

- Perangkat lunak SharIF Judge hanya digunakan pada lingkungan Teknik Informatika Unpar.
- Perangkat lunak hanya dapat diuji pada mata kuliah pemrograman di mana dosen pembimbing terlibat.

1.5 Metodologi

Metodologi pengerjaan tugas akhir ini adalah sebagai berikut:

1. Melakukan studi mengenai komponen yang diperlukan untuk membuat sistem perekaman dan pemutaran ulang ketikan pada IDE berbasis web.
2. Merancang sistem perekaman dan pemutaran ulang ketikan berbasis web untuk SharIF Judge
3. Mengimplementasikan IDE berbasis web pada SharIF Judge.
4. Melakukan pengujian dan eksperimen.
5. Menulis dokumen tugas akhir.

1.6 Sistematika Pembahasan

Sistematika pembahasan skripsi ini adalah sebagai berikut:

- **Bab 1:** Pendahuluan
  Membahas latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan.
- **Bab 2:** Landasan Teori
  Membahas teori-teori yang berhubungan dengan penelitian ini, yaitu SharIF Judge, CodeIgniter 3, Twig, IDE, dan Ace.
- **Bab 3:** Analisis
  Membahas analisis terhadap perangkat lunak SharIF Judge dan IDE pada SharIF Judge.
- **Bab 4:** Perancangan
  Membahas perancangan fitur yang diimplementasikan pada SharIF Judge.
- **Bab 5:** Implementasi dan Pengujian
  Membahas implementasi fitur pada SharIF Judge dan pengjuan yang dilakukan.
- **Bab 6:** Kesimpulan dan Saran
  Membahas kesimpulan dari penelitian ini dan saran untuk penelitian berikutnya.

BAB 2

LANDASAN TEORI

2.1 SharIF Judge

SharIF Judge merupakan modifikasi dari *open source* bernama Sharif Judge, sebuah website judge gratis dengan kemampuan mengkompilasi bahasa C, C++, Java, dan Python. Sharif Judge dibuat oleh Mohammad Javad Naderi dengan interface web berbahasa PHP menggunakan framework CodeIgniter dan BASH [1]. Modifikasi dilakukan untuk menambahkan fitur pada Sharif Judge dan juga untuk menyesuaikan sesuai dengan kebutuhan Teknik Informatika UNPAR.

2.1.1 Instalasi

Ada beberapa prasyarat yang diperlukan dalam menjalankan SharIF Judge pada sebuah *server* Linux adalah sebagai berikut:

- *Webserver* dengan PHP versi 5.3 atau lebih dengan `mysqli` extension
- PHP Command Line Interface (CLI)
- *Database* MySql atau PostgreSql
- PHP harus memiliki akses untuk menjalankan *shell commands* dengan fungsi `shell_exec`
- Kemampuan untuk mengompilasi dan menjalankan kode yang dikumpulkan (`gcc`, `g++`, `javac`, `java`, `python2`, dan `python3`)
- Perl

Setelah perangkat yang sudah memenuhi prasyarat, berikut merupakan cara instalasi SharIF Judge:

1. Unduh versi terakhir dari Sharif Judge dan menempatkannya pada direktori publik.
2. Pindahkan folder `system` dan `application` ke luar direktori publik. Kemudian simpan alamatnya pada `index.php`.
3. Buat sebuah *Database* MySql atau PostgreSql.
4. Atur pengaturan koneksi *database* pada `application/config/database.php`.
5. Atur pengaturan koneksi `RADIUS` dan `SMTP` pada `application/config/secrets.php` jika dibutuhkan.
6. Atur agar direktori `application/cache/Twig` dapat ditulis oleh php.
7. Buka halaman utama SharIF Judge pada *browser* dan ikuti proses instalasi.
8. Log in dengan akun admin
9. Pindahkan folder `tester` dan `assignments` ke luar direktori publik. Kemudian simpan alamatnya pada halaman pengaturan.

2.1.2 Users

Pada SharIF Judge, pengguna dibagi menjadi 4 buah *role*. Role yang tersedia adalah sebagai berikut:

1. *admin*
2. *head instructor*
3. *instructor*
4. *student*

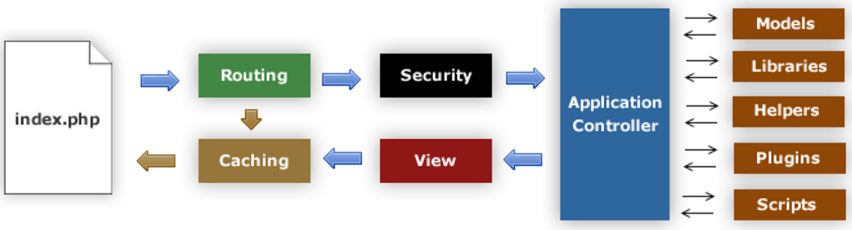
Setiap *role* memiliki akses pada aksi yang berbeda berdasakan *role*-nya. Tabel 2.1 merupakan aksi-aksi yang dapat dilakukan untuk setiap pengguna pada SharIF Judge.

Tabel 2.1: *Tabel fitur untuk setiap role*

Aksi	*Admin*	*Head Instructor*	*Instructor*	*Student*
Mengubah *Settings*	✓	×	×	×
Mengelola Pengguna	✓	×	×	×
Mengelola *Assignment*	✓	✓	×	×
Mengelola Notifikasi	✓	✓	×	×
Rejudge	✓	✓	×	×
Mengelola *Queue*	✓	✓	×	×
Mendeteksi Kode yang Mirip	✓	✓	×	×
Melihat Semua *Submission*	✓	✓	✓	×
Mengunduh Kode Final	✓	✓	✓	×
Memilih *Assignment*	✓	✓	✓	✓
Submit Kode	✓	✓	✓	✓

2.2 CodeIgniter 3

CodeIgniter 3 adalah sebuah *framework opensource* untuk mempermudah pengguna dalam membangun sebuah aplikasi *website* menggunakan bahasa PHP. CodeIgniter 3 bertujuan untuk membantu pengguna dalam membangun sebuah aplikasi *website* lebih cepat dengan menyedikan *library* yang beragam dengan fungsi yang umum digunakan dan tampilan dan *logic* yang simpel. Gambar 2.1 merupakan bagaimana data mengalir pada sistem CodeIgniter.



Gambar 2.1: *Flow Chart* CodeIgniter

Berikut merupakan penjelasan sederhana dari *flow chart* sistem CodeIgniter 3:

1. `index.php` berfungsi sebagai *front controller* yang akan melakukan inisiasi *resource* utama untuk menjalankan CodeIgniter.
2. Router meneliti *request* HTTP dan menentukan apa yang harus dilakukan dengan *request* tersebut.
3. Jika terdapat *file cache*, maka langsung dikirimkan ke *browser* melewati eksekusi sistem yang biasanya.

4. Sebelum *controller* dimuat, seluruh *request* HTTP dan data dari user disaring terlebih dahulu untuk keamanan.
5. *Controller* memuat *model*, *library* utama, dan *resource* lainnya yang diperlukan.
6. *View* akhir lalu dikirim ke browser untuk dilihat. *Cache* akan dibuat terlebih dahulu bila diaktifkan.

2.2.1 Model-View-Controller

CodeIgniter merupakan framework berbasis arsitektur Model-View-Controller atau yang selanjutnya akan disebut dengan MVC. MVC adalah pendekatan *software* yang memisahkan *logic* aplikasi dan tampilannya. Pendekatan ini membuat *website* hanya memiliki sedikit *script* karena tampilan *website* terpisah dari *scripting* PHP. Berikut merupakan penjelasan mengenai struktur MVC:

Model

Model mewakili struktur data pada sistem untuk mengambil, memasukkan, dan memperbaharui data pada *database*. *Model* dapat dibuat dengan membuat sebuah kelas yang mengekstensi `CI_Model` dan diletakkan pada `application/models/`.

Kode 2.1: Contoh *model*

```
class Blog_model extends CI_Model {

    public $title;
    public $content;
    public $date;

    public function get_last_ten_entries()
    {
            $query = $this->db->get('entries', 10);
            return $query->result();
    }

    public function insert_entry()
    {
            $this->title    = $_POST['title'];
            $this->content  = $_POST['content'];
            $this->date     = time();

            $this->db->insert('entries', $this);
    }

    public function update_entry()
    {
            $this->title    = $_POST['title'];
            $this->content  = $_POST['content'];
            $this->date     = time();

            $this->db->update('entries', $this, array('id' => $_POST['id']));
    }

}
```

Kode 2.1 merupakan contoh model kelas bernama `Blog_model` pada CodeIgniter. *Model* `Blog_model` dapat mengambil, menambahkan, dan mengperbaharui *database* bernama 'entries'. File *model* tersebut akan disimpan pada `application/models/Blog_model`. Selanjutnya, pengguna dapat memanggil *Model* tersebut pada *file controller* (akan dijelaskan pada bagian Controller) untuk memanggil model pada Kode 2.1 dengan mengunakan *syntax* sebagai berikut:

```
$this->load->model('Blog_model');.
```

Untuk memanggil *method* yang terdapat pada model tersebut, *syntax* yang digunakan adalah sebagai berikut:

```
$this->Blog_model->get_last_ten_entries();
```

Syntax diatas akan memuat `model` dengan nama `Blog_model` dan akan memanggil *method* `get_last_ten_entries`.

View

View adalah informasi yang akan di tunjukkan kepada user. Biasanya *view* merupakan sebuah halaman web, tetapi pada CodeIgniter, view dapat berupa pecahan halaman seperti *header, footer, sidebar*, dan lainnya. Pecahan halaman tersebut dapat dimasukkan secara fleksibel kedalam *view* lainnya apabila dibutuhkan.

Kode 2.2: Contoh *view*

```
<html>
<head>
        <title>My Blog</title>
</head>
<body>
        <h1>Welcome to my Blog!</h1>
</body>
</html>
```

Kode 2.2 merupakan contoh dari *file view* pada CodeIgniter. File akan disimpan pada direktori `application/views/`. Untuk dapat diperlihatkan dibutuhkannya penggalian halaman pada *file controller* dengan cara sebagai berikut:

```
$this->load->view('name');
```

Syntax diatas akan mengembalikan halaman *view* dengan nama `name` yang terletak pada direktori `application/views/name.php` dan menampilkannya kepada pengguna.

Controller

Controller adalah bagian utama dari aplikasi CodeIgniter, berfungsi sebagai perantara antara *model*, *view*, dan *resources* lainnya yang dibutuhkan untuk memproses HTTP *request* dan membuat sebuah web page. Kelas *Controller* akan mengekstensi `CI_Controller` dan disimpan pada `application/controllers/`. Contoh *controller* ditunjukkan pada Kode 2.3.

Kode 2.3: Contoh *controller*

```
<?php
class Blog extends CI_Controller {

        public function index()
        {
                echo 'Hello World!';
        }

        public function comments()
        {
                echo 'Look at this!';
        }
}
```

Kode 2.3 berfungsi dalam mengembalikan string sesuai dengan fungsi *controller* yang dipanggil. Nama file *controller* pada direktori `application/controllers/blog.php` dan metode diatas akan dijadikan segmen pada URL seperti berikut:

```
example.com/index.php/blog/index/
```

URL diatas akan mengembalikan sebuah teks 'Hello World!'.

Kode 2.4: Contoh memuat *model* dan menampilkan *view*

```
    class Blog_controller extends CI_Controller {

            public function blog()
            {
                    $this->load->model('blog');

                    $data['query'] = $this->blog->get_last_ten_entries();

                    $this->load->view('blog', $data);
            }
    }
}
```

Pada CodeIgniter, *model* dan *view* hanya dapat dimuat melalui controller. Seperti contoh, Kode 2.4 akan memuat *model* `blog` dan mengambil data dari *database*, lalu menampilkan *view* yang memuat data tersebut.

2.2.2 CodeIgniter URLs

URL pada CodeIgniter menggunakan *segment-based approach* dibandingkan dengan *query string approach* yang biasanya dipakai. *Segment-based approach* dirancang untuk *search-engine* dan dapat mempermudah pengguna juga. Berikut merupakan contoh dari URL CodeIgniter:

```
example.com/news/article/my_article
```

Struktur URL pada CodeIgniter juga mengikuti pendekatan MVC (referensi 2.2.1) dan biasanya memiliki struktur sebagai berikut:

```
example.com/class/function/ID
```

1. Segmen pertama mewakili kelas *controller* yang ingin dipanggil.
2. Segmen berikutnya mewakili fungsi kelas atau *method* yang ingin di panggil.
3. Segmen ketiga dan selanjutnya mewakili *identifier* atau pengenal dan variable-variable lain yang akan di kirimkan ke *controller*.

2.3 Twig

Twig merupakan sebuah *template engine* untuk PHP. Ada beberapa *expression*, *expression*, atau *statement* yang ditemupakan pada template Twig adalah sebagai berikut:

- Pewarisan *Template*
- Struktur Kontrol (mengunaan kondisional, *looping*)
- Filter
- Variable pada PHP

Pada saat template dievaluasi, semua *variable* atau *expression* akan dibuah menjadi value dan *tag* yang mengontrol logika template.

Kode 2.5: Contoh template Twig

```
{% extends "base.html" %}
{% block navigation %}
    <ul id="navigation">
    {% for item in navigation %}
        <li>
            <a href="{{item.href}}">
                {% if item.level == 2 %}  {% endif %}
                {{ item.caption|upper }}
            </a>
        </li>
    {% endfor %}
    </ul>
{% endblock navigation %}
```

Kode 2.5 merupakan contoh sebuah template Twig. Terdapat dua jenis *delimiter*, yaitu `{% ... %}` dan `{{ ... }}`. *Delimiter* `{% ... %}` digunakan untuk Menjalankan sebuah *statement* seperti *for-loops*, sedangkan *delimiter* `{{ ... }}` digunakan untuk mengubah sebuah *variable* atau *expression* menjadi nilai sesungguhnya.

2.4 Integrated Development Environment

Intergrated Development Environment (IDE) merupakan sebuah aplikasi yang menyediakan berbagai peralatan yang diperlukan untuk membantu pengembangan perangkat lunak. Beberapa peralatan umum yang dimiliki oleh sebuah IDE adalah sebagai berikut:

- *Editor*
  Editor teks sebagai tempat untuk mengetik kode, dapat dilengkapi dengan berbagai fitur seperti *syntax highlighting* (menampilkan teks dengan warna yang berbeda untuk menginkatkan keterbacaan kode) dan *word completion* (menampilkan prediksi kata yang sedang atau yang akan diketik pengguna).

- *Complier*
  Digunakan untuk menterjemahkan kode program yang dibuat pada editor teks kedalam sebuah program yang dapat dijalankan oleh komputer.
- *Execution*
  Menjalankan kode program yang sudah dikompilasi, dengan input jika dibutuhkan, dan mengembalikan hasilnya.

2.5 Ace

Ace merupakan sebuah editor kode yang dapat dimasukkan kedalam sebuah website yang dibuah mengunakan bahasa *Javasciprt*. Ace memiliki kemampuan dari editor pada umumnya. Berikut merupakan beberapa fitur utama yang dimiliki oleh Ace:

- *Syntax highlighting* untuk bahasa pemrograman.
- Automatic indent dan outdent.
- Kemampuan *cut*, *copy*, dan *paste*.
- Kemampuan *drag and drop* teks menggunakan mouse.
- Banyak *Cursors* dan *selections*
- *Line wrapping*
- *Code folding*

Beberapa kelas penting yang terdapat pada library Ace adalah sebagai berikut:

- `Ace`
  Merupakan kelas utama untuk menyiapkan editor kode Ace pada *browser*
- `Anchor`
  Menangani posisi *pointer* pada dokumen.
- `Document`
  Menyimpan teks dokumen.
- `Editor`
  Entri utama untuk fungsionalitas library Ace.

Kode 2.6: Contoh kode pengunaan Ace

```
<!DOCTYPE html>
<html lang="en">
<head>
<title>ACE in Action</title>
<style type="text/css" media="screen">
    #editor {
        position: absolute;
        top: 0;
        right: 0;
        bottom: 0;
        left: 0;
    }
</style>
</head>
<body>

<div id="editor">function foo(items) {
    var x = "All this is syntax highlighted";
    return x;
}</div>

<script src="/ace-builds/src-noconflict/ace.js" type="text/javascript" charset="utf-8"></script>
<script>
    var editor = ace.edit("editor");
    editor.setTheme("ace/theme/monokai");
    editor.session.setMode("ace/mode/javascript");
</script>
</body>
</html>
```

Kode 2.6 merupakan cara pengunaan Ace pada sebuah `div` dengan id `editor`. Ace juga memiliki beberapa konfigurasi, seperti contoh ini yaitu mengunakan tema *monokai* dan mengunakan *syntax highlighting* untuk bahasa pemrograman JavaScript.

DAFTAR REFERENSI

[1] Version 1.4 (2014) *Sharif Judge Documentation.* Mohammad Javad Naderi. Tehran, Iran.

[2] Vallian, S. (2018) Kustomisasi Sharif Judge Untuk Kebutuhan Program Studi Teknik Informatika. Skripsi. Universitas Katolik Parahyangan, Indonesia.

[3] Halim, N. A. (2021) Implementasi Editor Kode pada SharIF Judge. Skripsi. Universitas Katolik Parahyangan, Indonesia.

[4] Version 1.4.13 (2021) *Ace API Reference.* Ajax.org B.V. Amsterdam, The Netherlands.

LAMPIRAN A

KODE PROGRAM

Kode A.1: MyCode.c

```c

// This does not make algorithmic sense,
// but it shows off significant programming characters.

#include<stdio.h>

void myFunction( int input, float* output ) {
    switch ( array[i] ) {
        case 1: // This is silly code
            if ( a >= 0 || b <= 3 && c != x )
                *output += 0.005 + 20050;
            char = 'g';
            b = 2^n + ~right_size - leftSize * MAX_SIZE;
            c = (--aaa + &daa) / (bbb++ - ccc % 2 );
            strcpy(a,"hello $@?");
    }
    count = ~mask | 0x00FF00AA;
}

// Fonts for Displaying Program Code in LATEX
// Adrian P. Robson, nepsweb.co.uk
// 8 October 2012
// http://nepsweb.co.uk/docs/progfonts.pdf
```

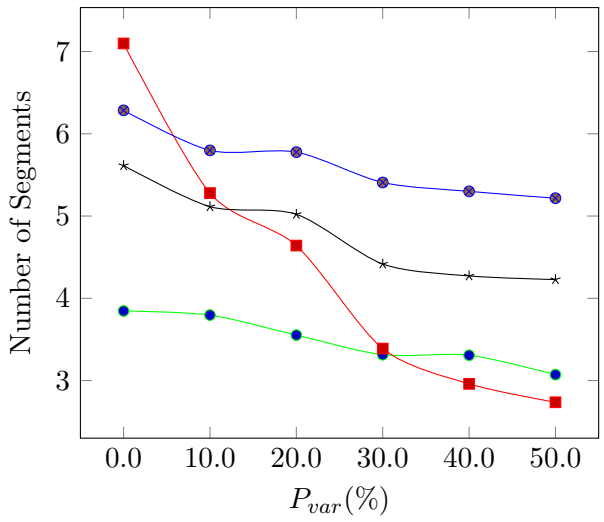
Kode A.2: MyCode.java

```java
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashSet;

//class for set of vertices close to furthest edge
public class MyFurSet {
    protected int id;                                   //id of the set
    protected MyEdge FurthestEdge;                      //the furthest edge
    protected HashSet<MyVertex> set;                    //set of vertices close to furthest edge
    protected ArrayList<ArrayList<Integer>> ordered;    //list of all vertices in the set for each trajectory
    protected ArrayList<Integer> closeID;               //store the ID of all vertices
    protected ArrayList<Double> closeDist;              //store the distance of all vertices
    protected int totaltrj;                             //total trajectories in the set

    /*
     * Constructor
     * @param id : id of the set
     * @param totaltrj : total number of trajectories in the set
     * @param FurthestEdge : the furthest edge
     */
    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
        this.id = id;
        this.totaltrj = totaltrj;
        this.FurthestEdge = FurthestEdge;
        set = new HashSet<MyVertex>();
        ordered = new ArrayList<ArrayList<Integer>>();
        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
        closeID = new ArrayList<Integer>(totaltrj);
        closeDist = new ArrayList<Double>(totaltrj);
        for (int i = 0;i <totaltrj;i++) {
            closeID.add(-1);
            closeDist.add(Double.MAX_VALUE);
        }
    }

}
```

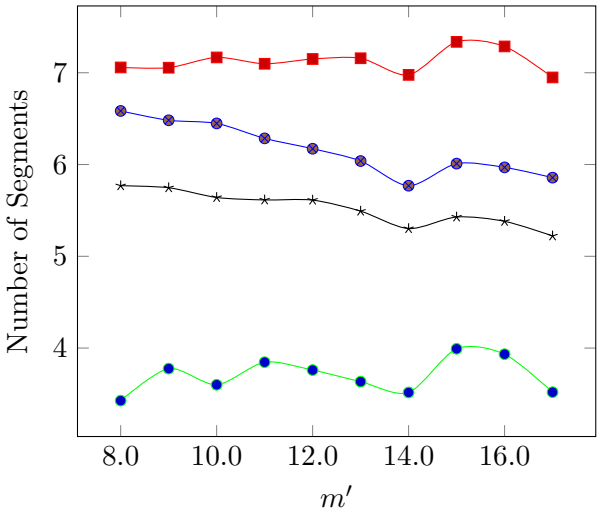
LAMPIRAN B

HASIL EKSPERIMEN

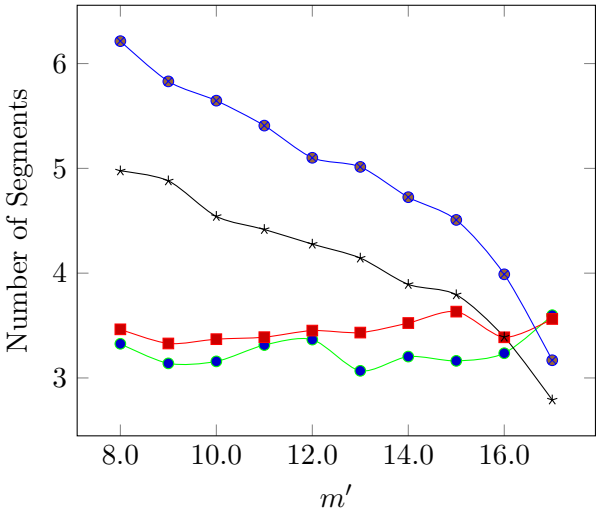
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



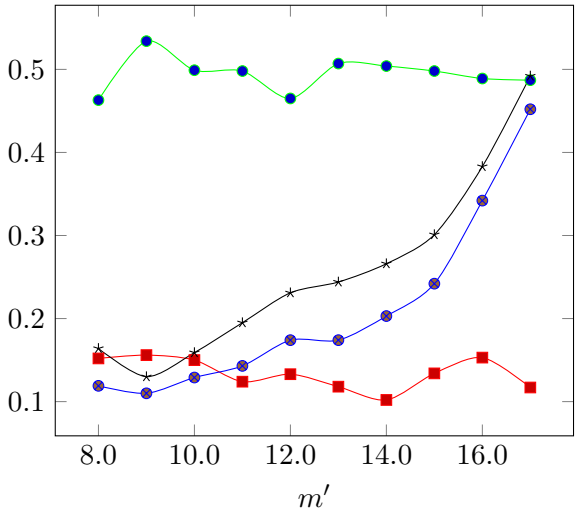
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4