Final Project Report

CSE5331: Advance Algorithms  
Plagio Plagiot : Plagiarism Detector

Dec 18, 2015

Implementation of various string algorithms

And their Execution Time Comparison

Team

Atul Konaje, 1001145198

Samvaran Kashyap Rallabandi 1001142545

Yamini Sai Lakshmi Jagarapu, 1001153306

**Table of Contents**

[1 Introduction 3](#_Toc438232966)

[2 Technologies Used 3](#_Toc438232967)

[3 Architecture 3](#_Toc438232968)

[4 Project File structure: 4](#_Toc438232969)

[5 Commands to run the application 5](#_Toc438232970)

[6 Features 6](#_Toc438232971)

[7 Project Evaluation 6](#_Toc438232972)

[8 Project Contributions 11](#_Toc438232973)

[9 Conclusions of Evaluation 11](#_Toc438232974)

[10 Screenshots 11](#_Toc438232975)

[11 Webbilography 14](#_Toc438232976)

# Introduction

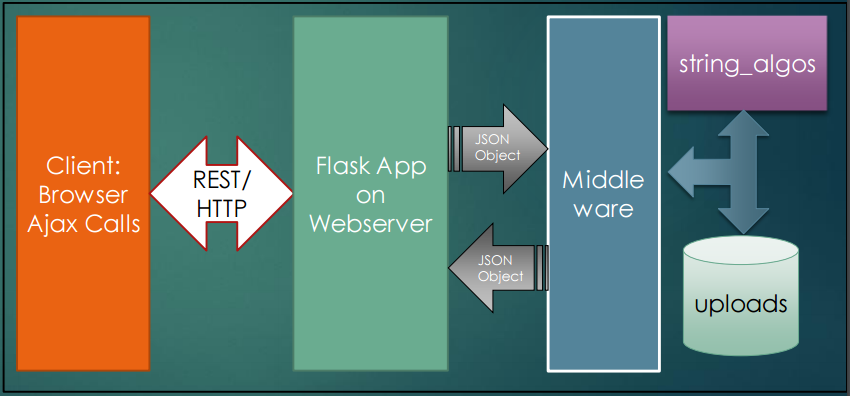
Plagio Plagiot is a Web based application developed in vision to detect the plagiarism detection of large corpus files , Further it also emphasizes on the extensive evaluation of the various string matching algorithms . End users of the given application will be able to upload their own plain text files to the server and run various string algorithms on the set of files to detect the various string matches and subsequence matches. Plagio Plagiot has a dynamic AJAX query based user interface which makes it loosely coupled with the server side RESTful APIs. The application also involves in the implementation of object factory pattern in middleware, making it extensibvle for the future use . However , there is a constraint in the implementation of the string algorithms which is the object oriented implementation of the algorithms should follow a specific json object format as a return value. Currently the files are stored in a flat file system of the server side, which can also be configured to the database.

# Technologies Used

Being a Web Based tool the current application used both server side and Client Side technologies as follows:

* **Client Side :**
  + **Jquey 1.9.1:**
    - Used for implementing Tabbed interface
    - Used for the making AJAX calls to the serverside RESTFUL APIs
  + **BootStrap 3.0:**
    - Used for the styling of the existing webpage.
* **Server Side :**
  + **Python 2.7:**
    - Programming language is chosen for the server side implementation.
  + **Flask Framework:**
    - For implementing a RESTful Services.

# Architecture



1. **Client Side:**

The client side is implemented in Jquery 1.9.1 and HTML , where the whole requests are being made to the server side scripting Restful services. The response of REST API services are being formatted into a simple html table. .   
 The userinteface is implemented by JQUERY tabs which consists of three tabs each for uploading files, running user selected algorithm on files and patterns respectively.

1. **Communication:**

The whole communication between the client side Jquery library functions and the flask REST API happens in HTTP.

1. **FlaskApp On Webserver:**

Flask acts REST service provider which provides following services :

* / : responsible for rendering homepage and handling the upload files
* /getfiles : provides all the files on the uploads folder
* /plagcheck : Plagcheck takes two filenames(pattern file, corpus file/files), algorithm to be ran as input and outputs the postions/sequences where there is a match
* /patterncheck : Plagcheck takes two strings(pattern , text ), algorithm to be ran as input and outputs the postions/sequences where there is a match.

1. **Middleware:**

Middleware is responsible for the request forwarding from the flaskapp server to the appropriate algorithm in string algos package

1. **String Algos Package:**

The string algos package is an object oriented implementation of naïve string search , Knuth morris pattern , Boyermore moore , longest common subsequence algorithm.

1. **Uploads:**

Uploads folder is a flat file system storage where uploaded files are being stored. It can be configured in the index.py file.

# Project File structure:

plagiarism\_detector

├── BoyerMore\_client.py

├── index.py

├── KMP\_client.py

├── LCSS\_client.py

├── middleware.py

├── middleware.pyc

├── NaiveSearch\_client.py

├── README.md

├── static

│   ├── css

│   │   ├── bootstrap.min.css

│   │   └── bootstrap-theme.min.css

│   └── js

│   ├── bootstrap.min.js

│   └── jquery.js

├── string\_algos

│   ├── BoyerMore.py

│   ├── BoyerMore.pyc

│   ├── \_\_init\_\_.py

│   ├── \_\_init\_\_.pyc

│   ├── KMP.py

│   ├── KMP.pyc

│   ├── LCSS.py

│   ├── LCSS.pyc

│   ├── NaiveSearch.py

│   └── NaiveSearch.pyc

├── templates

│   └── index.html

└── uploads

├── check.txt

├── copykingkhyber2.txt

├── copykingofkhyber1.txt

├── corpustext1.txt

├── kingofkhyber.txt

├── pg13316.txt

├── pg16712.txt

├── pg5246.txt

├── pg7514.txt

├── plagtext1.txt

├── plagtext2.txt

├── TheChild\_100\_Line\_Pattern.txt

├── TheChild\_100Line\_Pattern.txt

├── TheChild\_10\_Line\_Pattern.txt

├── TheChild\_150\_Line\_Pattern.txt

├── TheChild\_1Line\_Pattern.txt

├── TheChild\_200Line\_Pattern .txt

├── TheChild\_248Line\_Pattern.txt

├── TheChild\_25\_Line\_Pattern.txt

├── TheChild\_2Line\_Pattern.txt

├── TheChild\_35Line\_Pattern.txt

├── TheChild\_50\_Line\_Pattern.txt

├── TheChild\_NLine\_Pattern.txt

├── TheChildrenOfForest.txt

├── TheChildrenofNewForest\_1000.txt

├── TheChildrenofNewForest\_250.txt

├── TheChildrenofNewForest\_5000.txt

├── TheChildrenofNewForest\_500.txt

├── TheChildrenofNewForest.txt

└── TheChildren.txt

6 directories, 53 files

# Commands to run the application

* pip install flask #flask package needs to be installed in the systems
* git clone <https://github.com/samvarankashyap/plagiarism_detector>
* cd plagiarism\_detector
* python index.py
* # visit localhost:8088 in browser

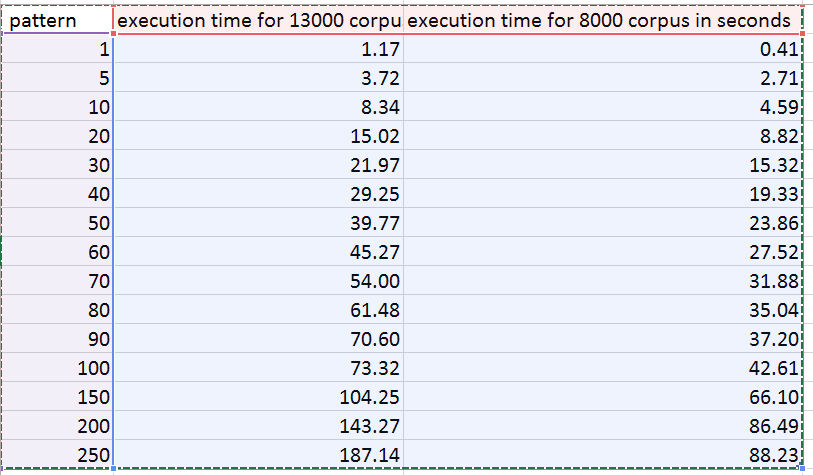
# Features

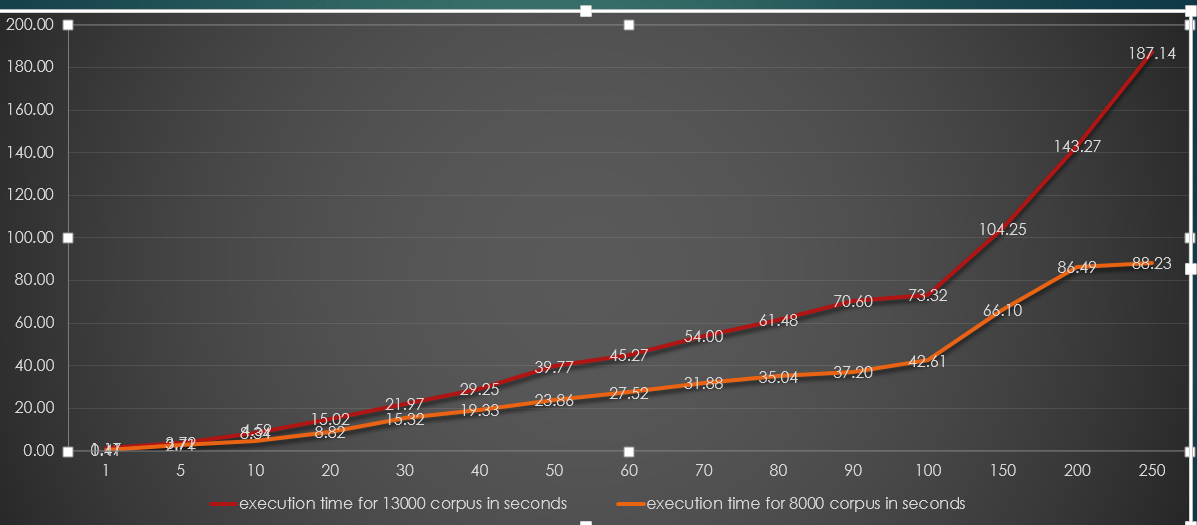
* User can Upload Files
* User can Run Pattern Match on Individual Strings or Files
* User can Select Among the four Algorithms
  + Naïve String Search
  + LCSS
  + KMP
  + Boyer More Horspool Algorithm

# Project Evaluation

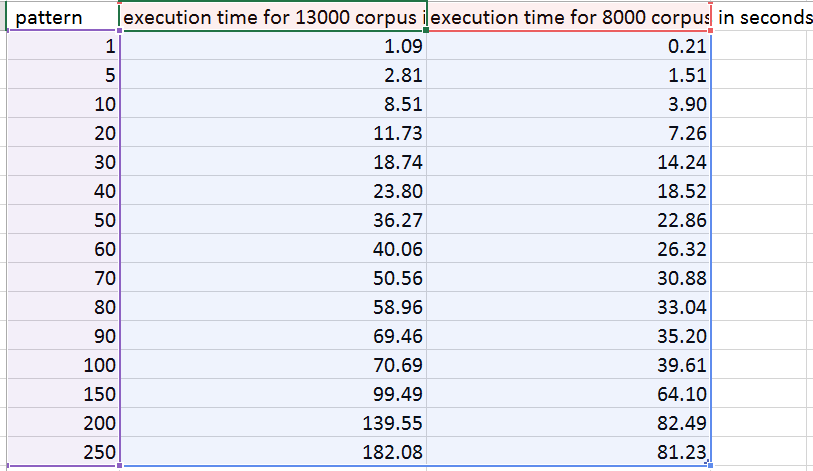
The current pictures depict the performance analysis of all the algorithms against different sizes datasets:

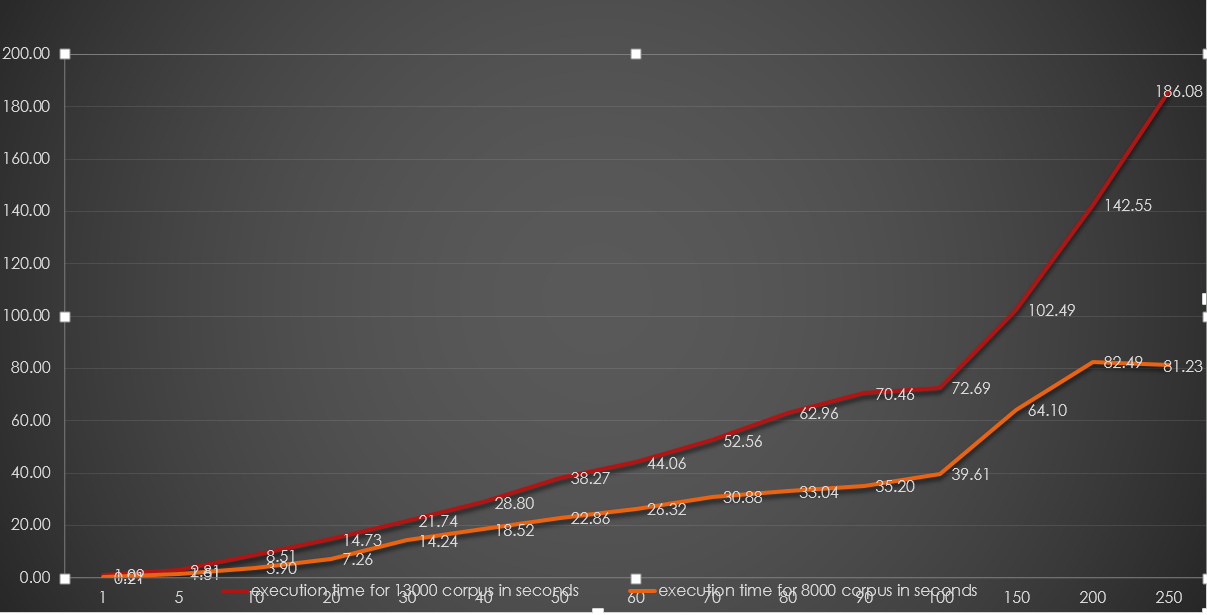
**Evaluation: Algorithm : Naïve string Search**



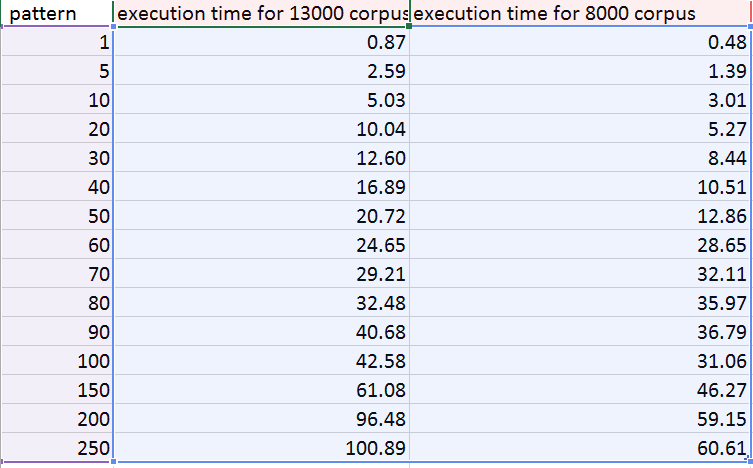


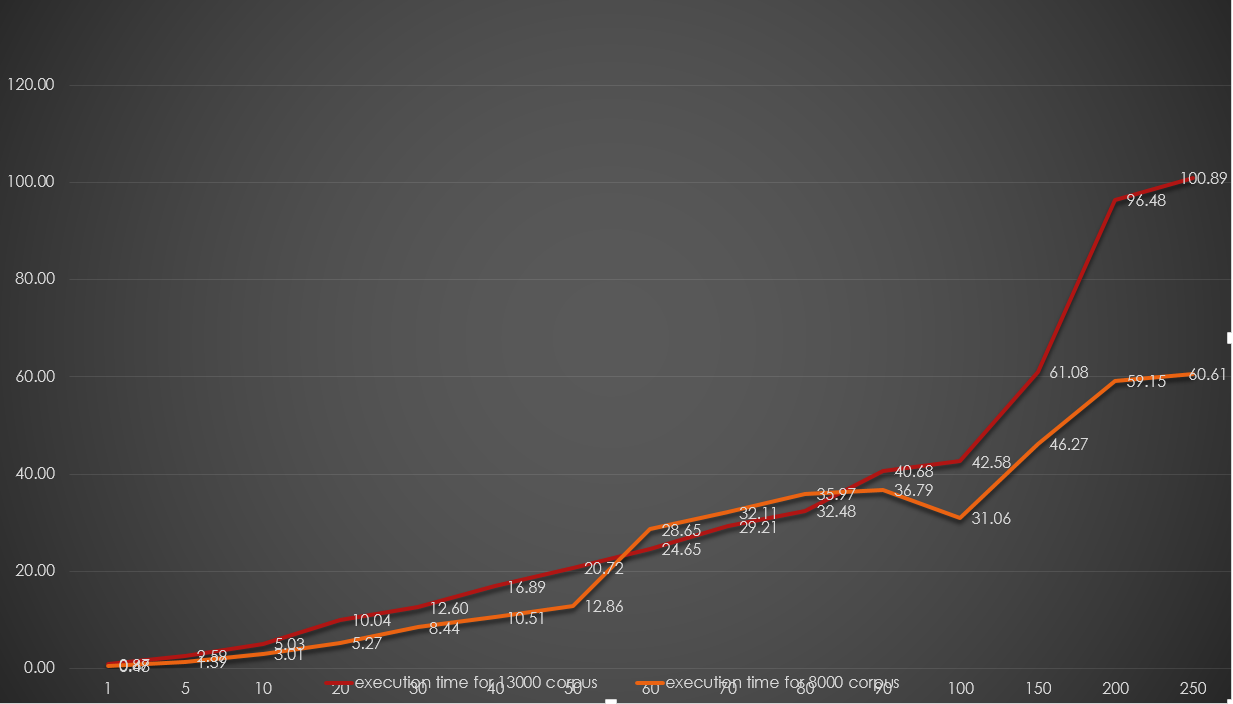
**Evaluation: Algorithm : kmp**



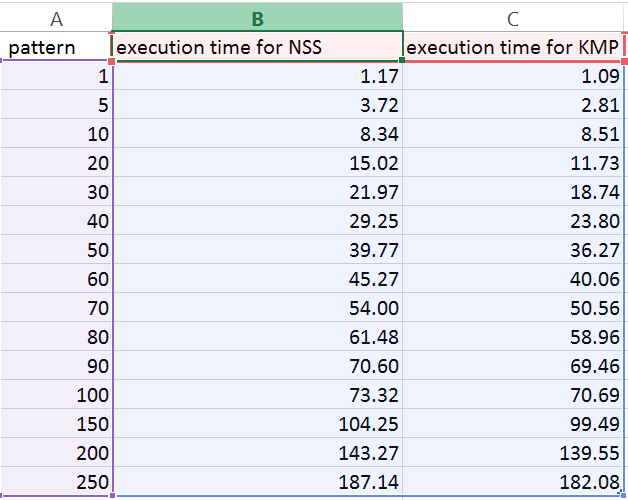


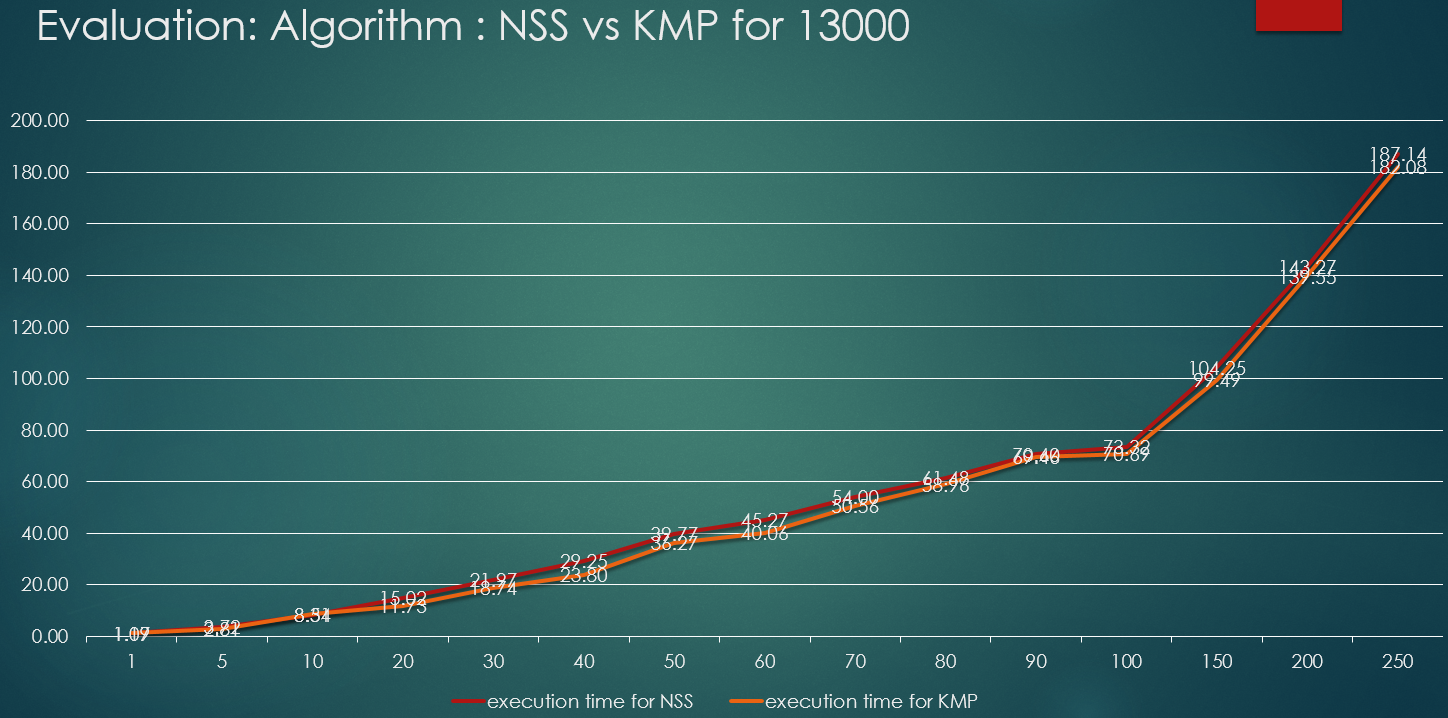
**Evaluation: Boyer moore:**

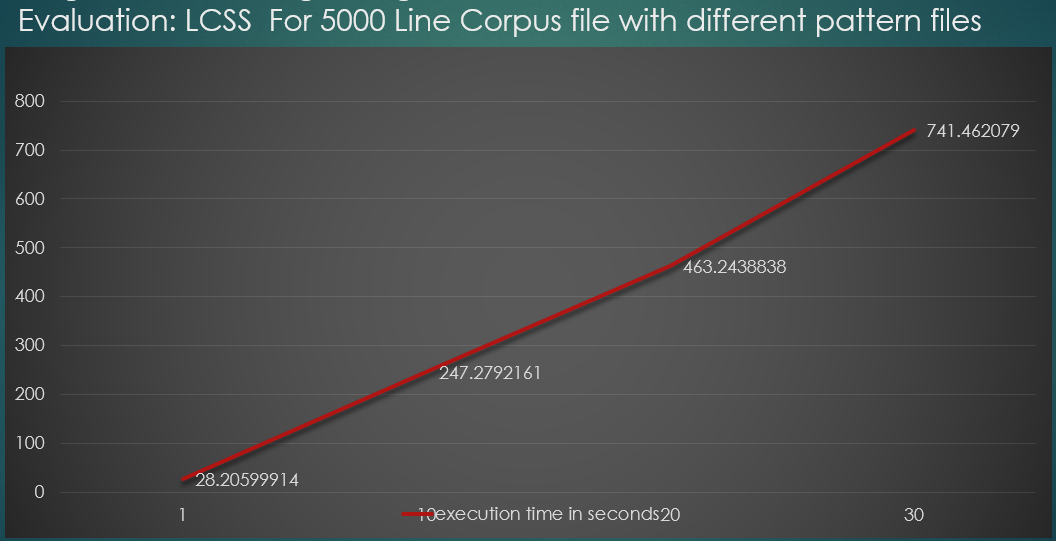


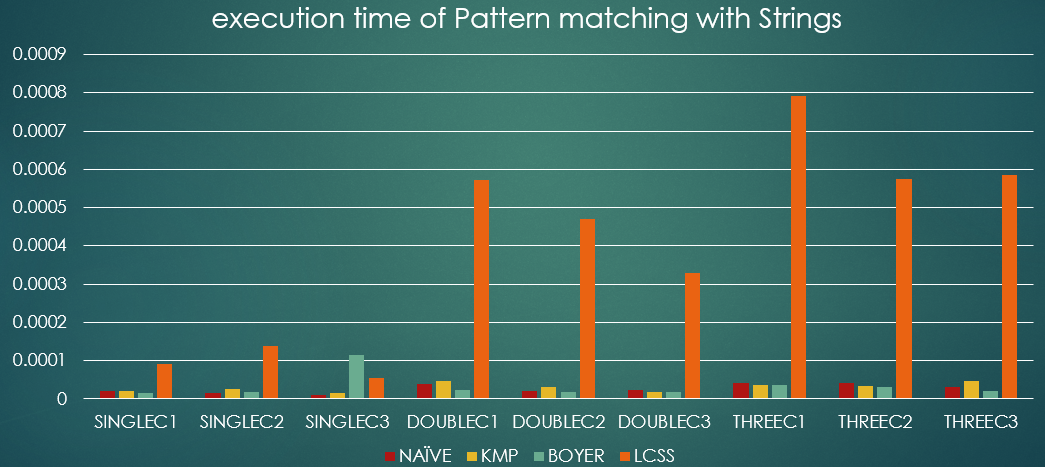


**Evaluation: NSS vs KMP for 13000**

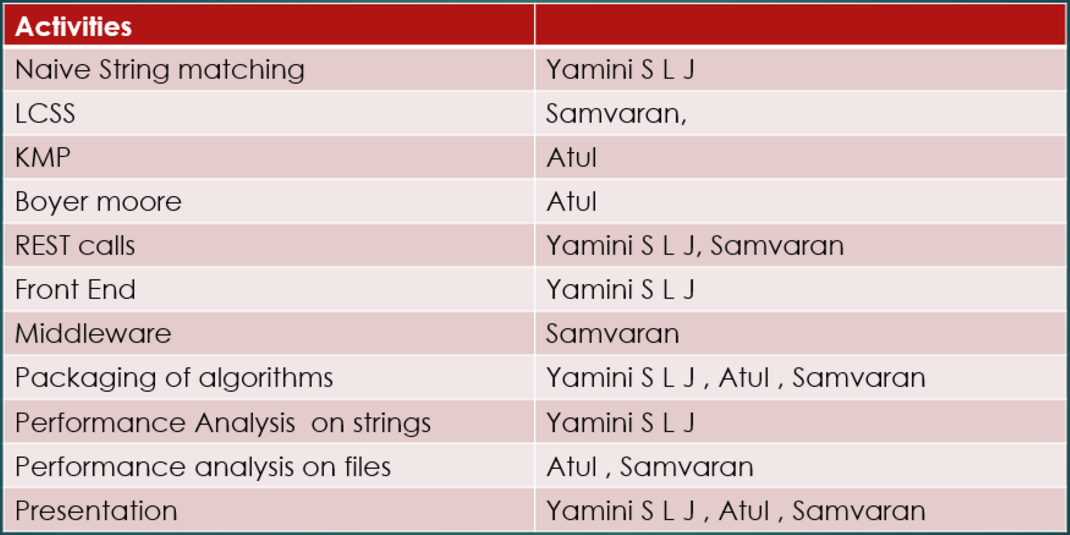








# Project Contributions



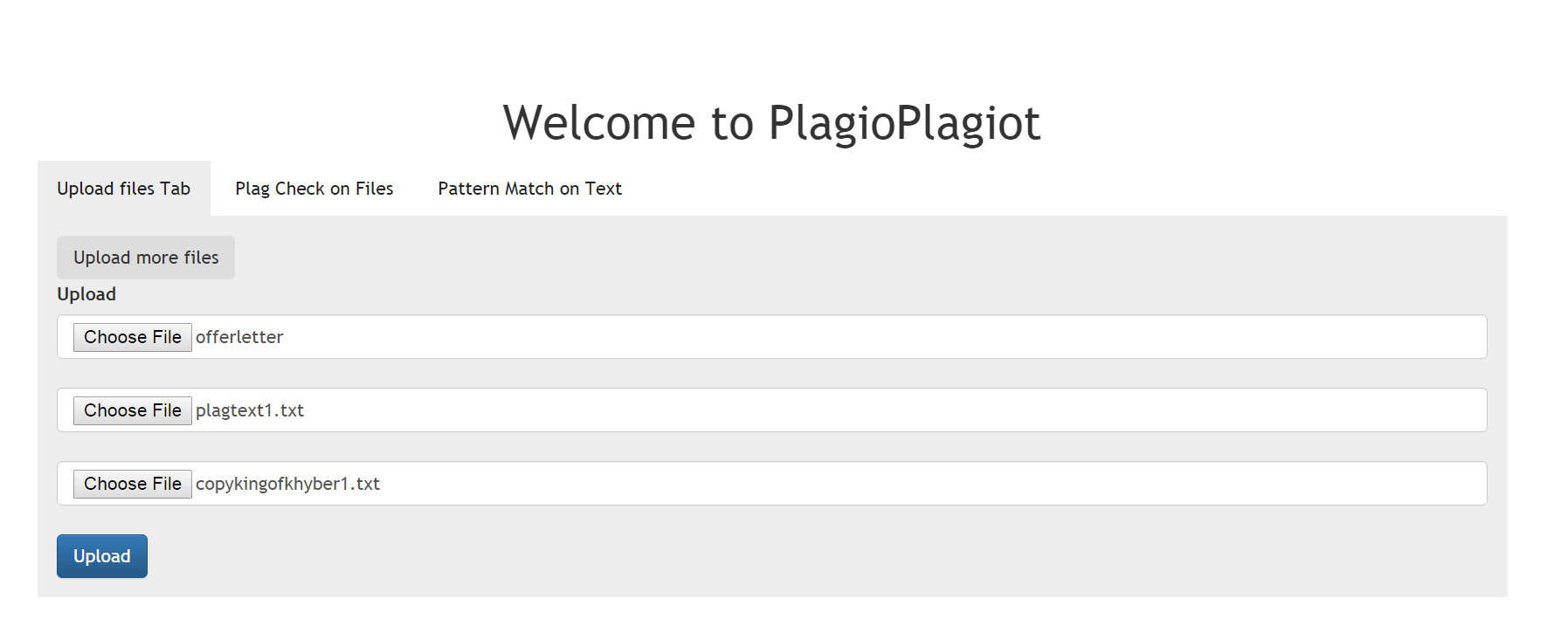
# Conclusions of Evaluation

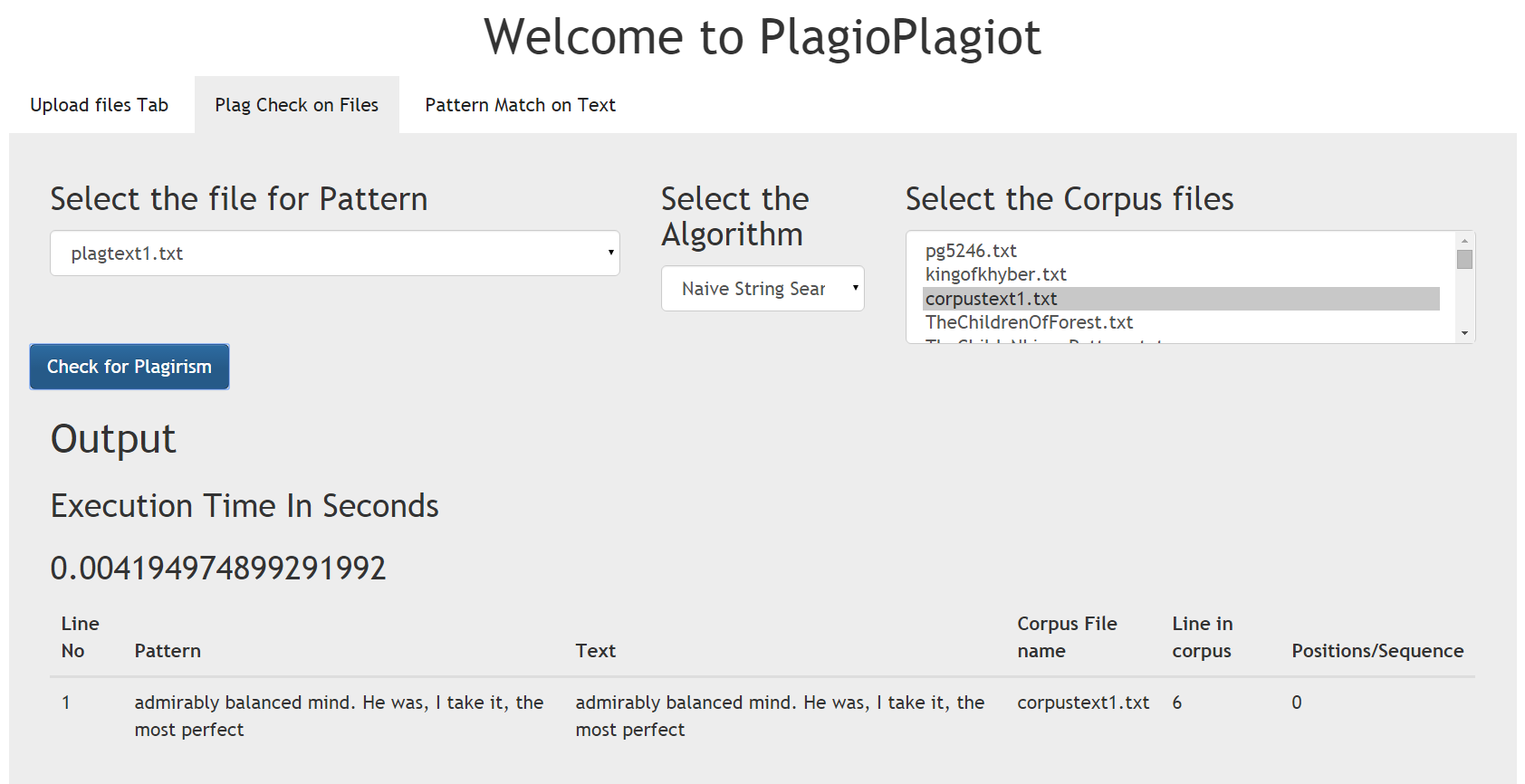
According to the evaluation of the project KMP algorithm is proved to be better than the Naïve string search. However , Boyer more has a best performance among all the algorithms. LCSS algorithm tries to find the longest comman subsequence with the given inputs results are taking more execution time than the other algorithms implemented.

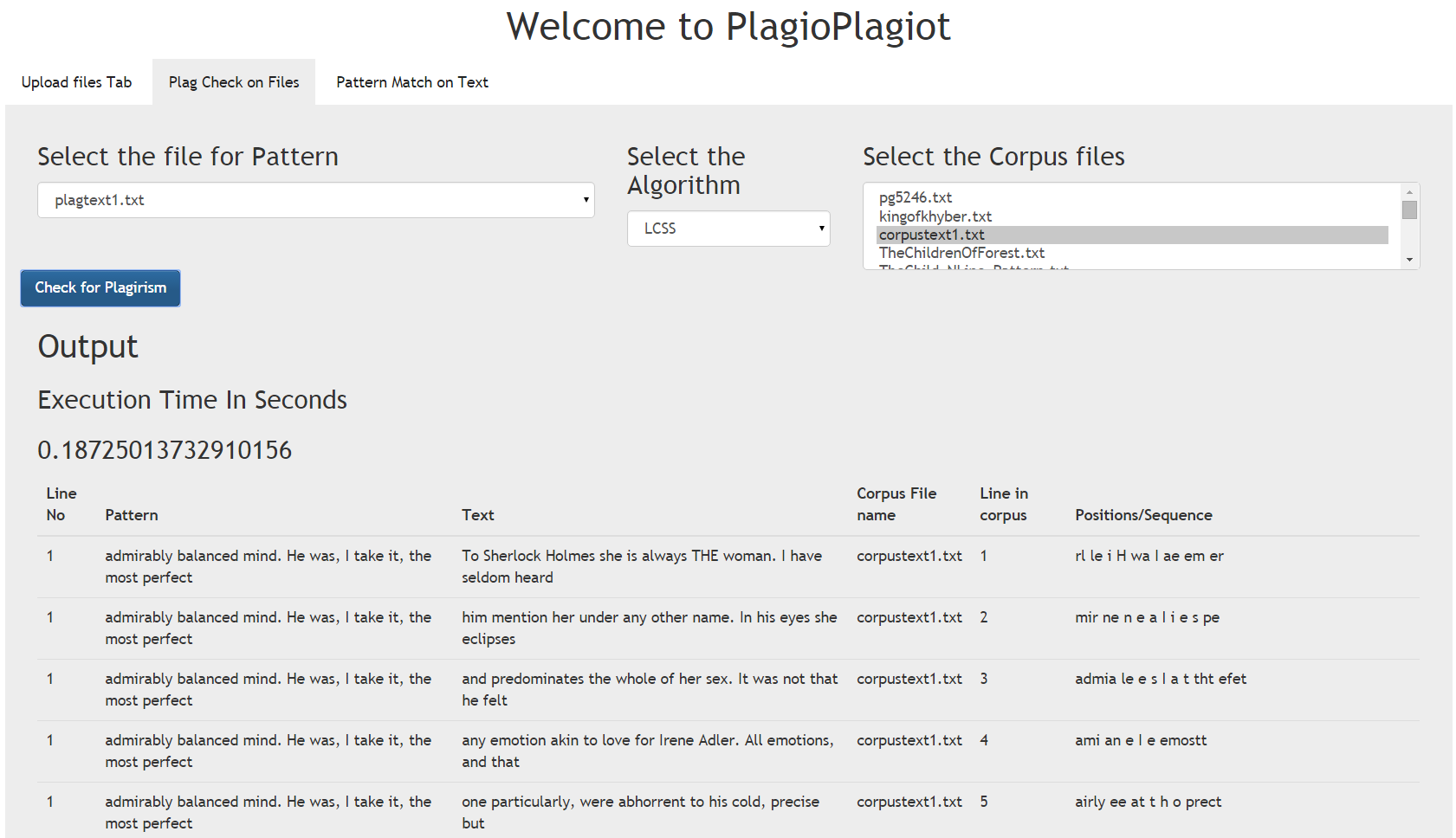
**Project code is available at following url :**<https://github.com/samvarankashyap/plagiarism_detector>

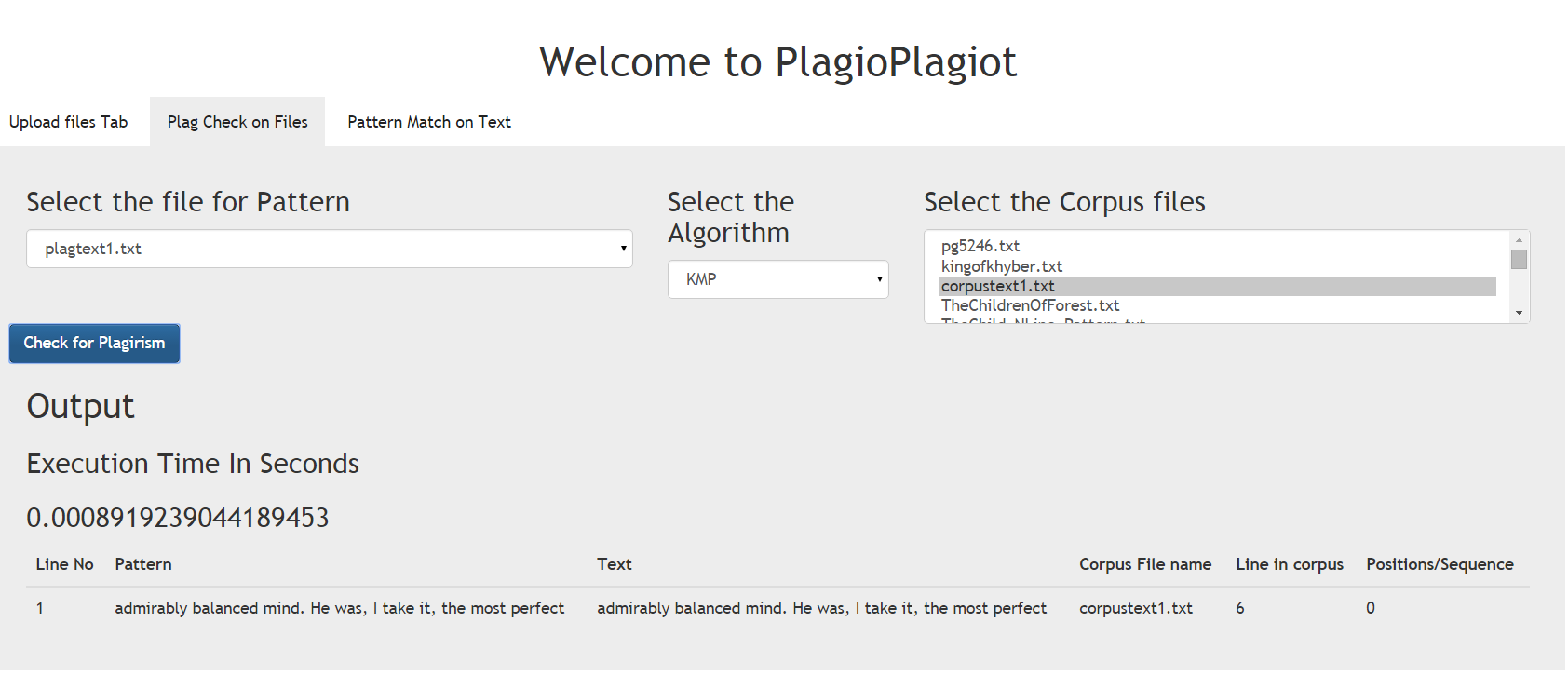
# Screenshots

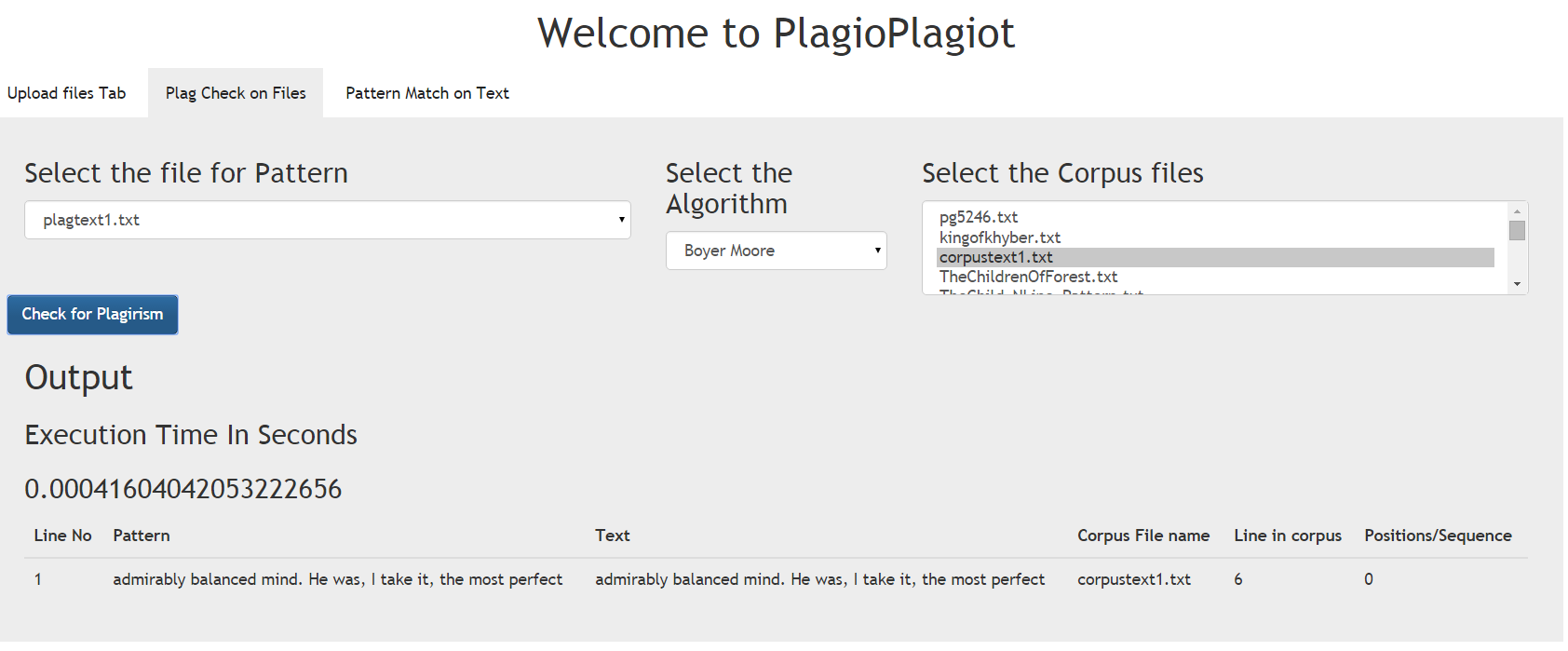
Aplication screenshots are as follows :

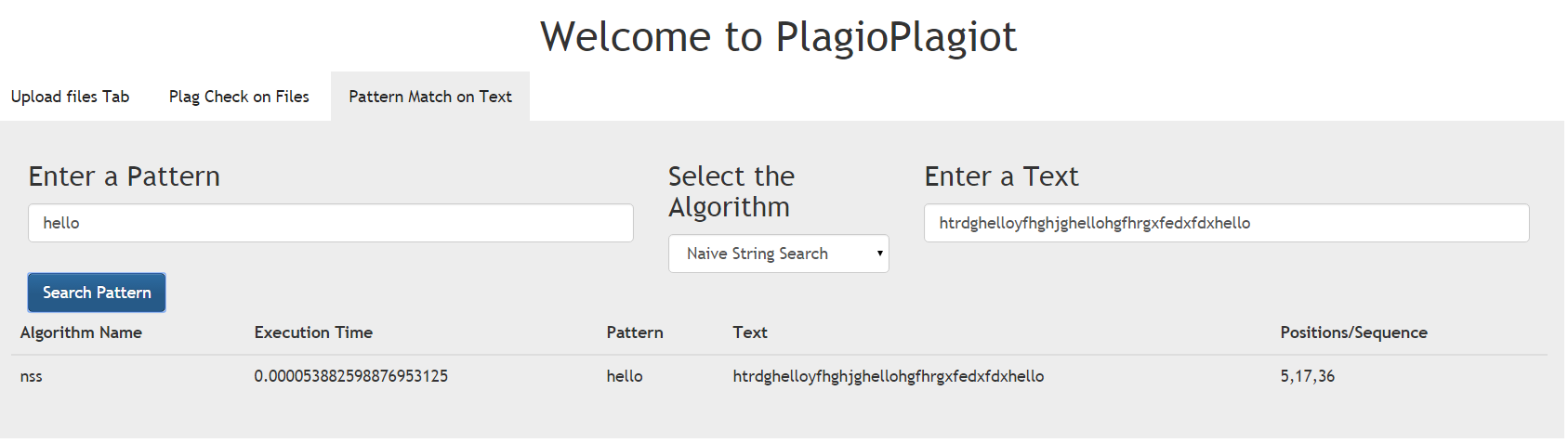


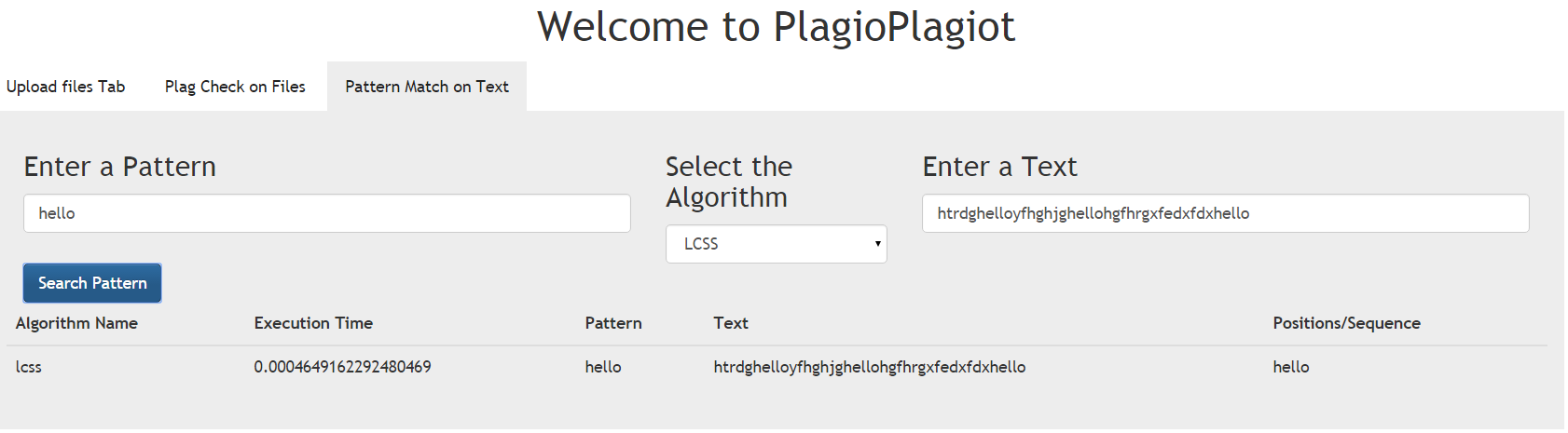


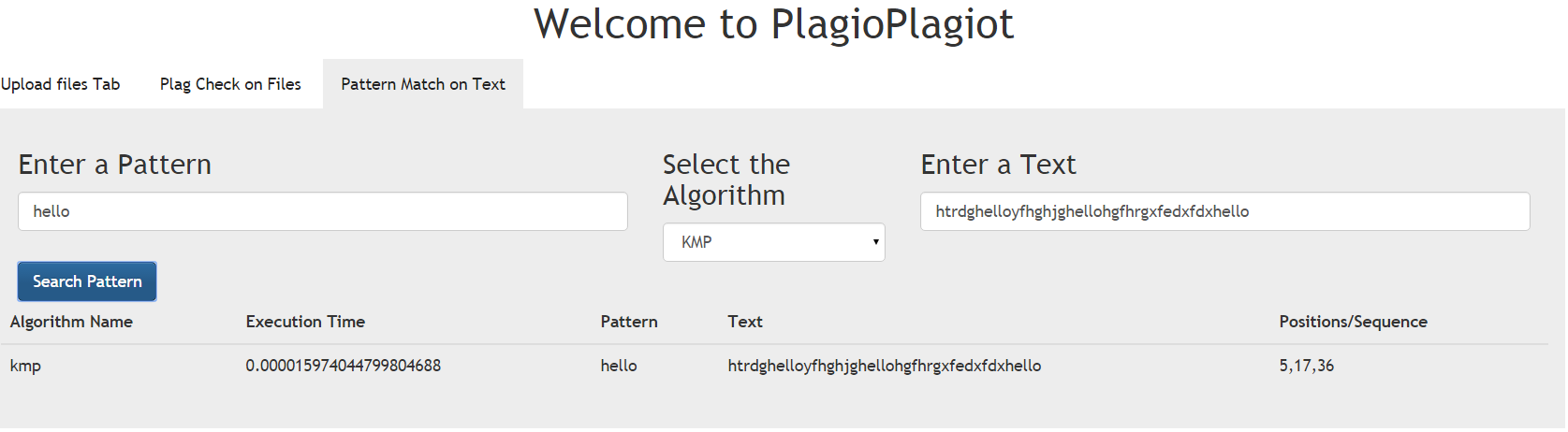


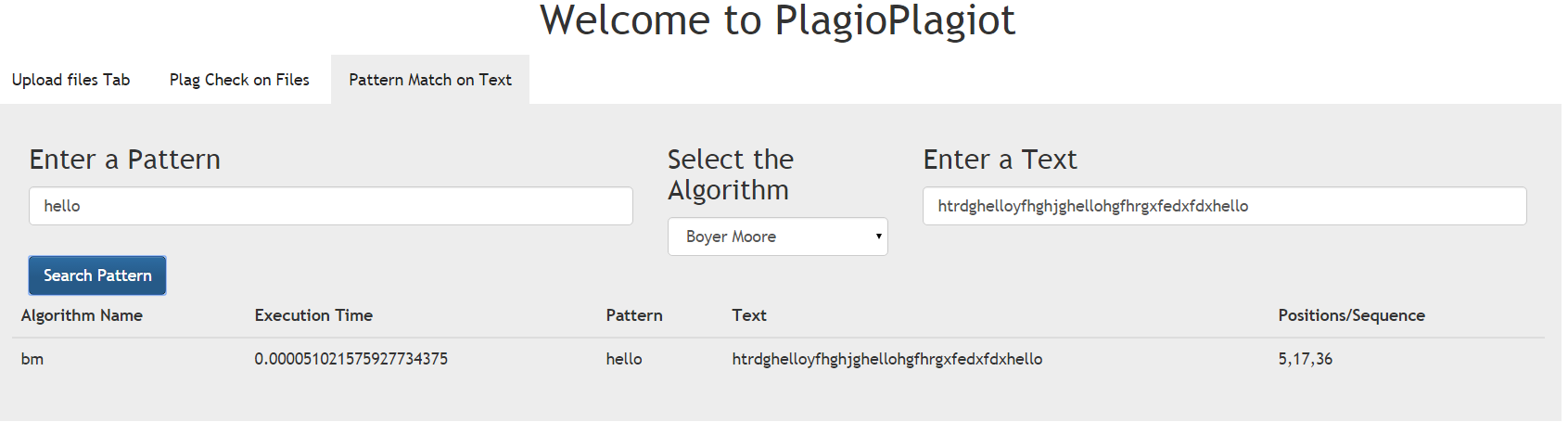












# Webbilography

[1] http://api.jquery.com/jquery.ajax/

[2] http://flask.pocoo.org/docs/0.10/quickstart/

[3] <https://en.wikipedia.org/wiki/Boyer%E2%80%93Moore_string_search_algorithm>

[4] <https://en.wikipedia.org/wiki/Knuth%E2%80%93Morris%E2%80%93Pratt_algorithm>

[5] http://www.geeksforgeeks.org/searching-for-patterns-set-1-naive-pattern-searching/

[6] http://blog.miguelgrinberg.com/post/designing-a-restful-api-with-python-and-flask

[7] <https://en.wikipedia.org/wiki/Longest_common_subsequence_problem>

[8] <http://getbootstrap.com/getting-started/>