**Final Project Report**

**CSE 5311: Design and Analysis of Algorithms**

**String Matching: Plagiarism detection**

**Implementation of different string matching**

**Algorithms and their running time analysis**

**Team:**

Krishna Chaitanya Rudraraju, 1001225036

Samson Yerraguntla, 1001234836

Siri Varma Vegiraju, 1001307644

**Table of Contents**

[1 Introduction 1](#_Toc438232966)

[2 Technologies Used 1](#_Toc438232967)

[3 Architecture 2](#_Toc438232968)

[4 Features 3](#_Toc438232971)

[5 Project Evaluation 4](#_Toc438232972)

[6 Project Contributions 6](#_Toc438232973)

[7 Conclusions of Evaluation 7](#_Toc438232974)

[8 Screenshots 7](#_Toc438232975)

[9 Webbilography 9](#_Toc438232976)

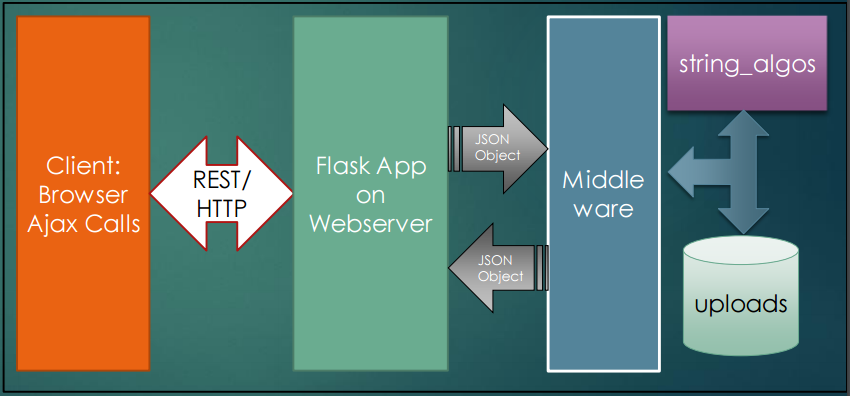
**1. Introduction**

Plagiarism detector is a web application that has string matching algorithm as its back bone. The tool is used to detect plagiarism for large corpus files and also for individual strings and patterns. The matched string and the position of the match found is displayed. The time analysis of the input algorithms on various different input files or strings is displayed as a time graph and also the time graph analysis of the input files and all the four algorithms is displayed. The web page communicates with back end using AJAX calls which make it loosely coupled with the server side RESTful API’s. As we select the required algorithm, a respective object is created and the algorithm is called. All the unnecessary data is filtered by the middleware. Uploads folder contains the input string files that are fed as input to the string matching algorithms. Facilitating the tool for future use, object factory pattern is implemented in the middleware

**2. Technologies Used**

* **Client Side:**
  + **Bootstrap 8.0:**
    - Used for styling and for developing automatically adjustable pages for various screens.
  + **JQuery 1.9.1**
    - Used for communications with back end using AJAX calls to the server side RESTful API’s
* **Server side:**
  + **Python 2.7:**
    - Used for Server side programming
  + **Flask Framework:**
    - Used for RESTful API’s.

**3. Architecture**



1. **Client Side:**

The front end is implemented using JQuery, HTML and Bootstrap. The server side scripting RESTful services accept all the requests.

There are three tables which enable us to toggle between uploading files, giving input strings and running and time analysis of the selected algorithms.

1. **Communication:**

HTTP enables communication between the flask REST API’s and client side JQuery library functions.

1. **FlaskApp On Web server:**

Flask is the web framework used for the application:

/ renders the homepage which has the all the functionalities of the app.

/getfiles: helps the user to upload files for plagiarism checker

/plagcheck: Plagcheck takes two filenames (pattern file, corpus file) algorithm to be ran as input and outputs the positions/strings matched.

/patterncheck: Plagcheck takes two strings (pattern, text), algorithm to be ran as input and outputs the positions

1. **Middleware:**

Middleware responsible for the requests made from the flaskapp server to run the application.

1. **String Algos Package:**

The string algos package has the implementations of Naïve String Search, Knuth Morris Pattern, Boyer Moore, Longest Common Subsequence algorithm.

1. **Uploads:**

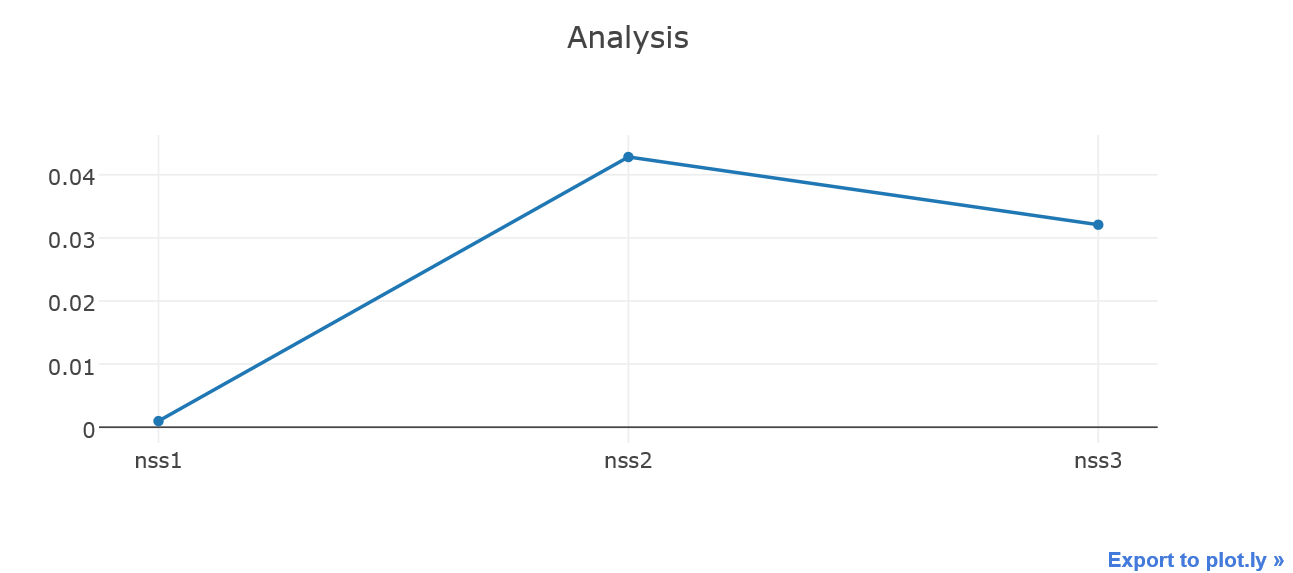
Contains all the files uploaded from the local directory.

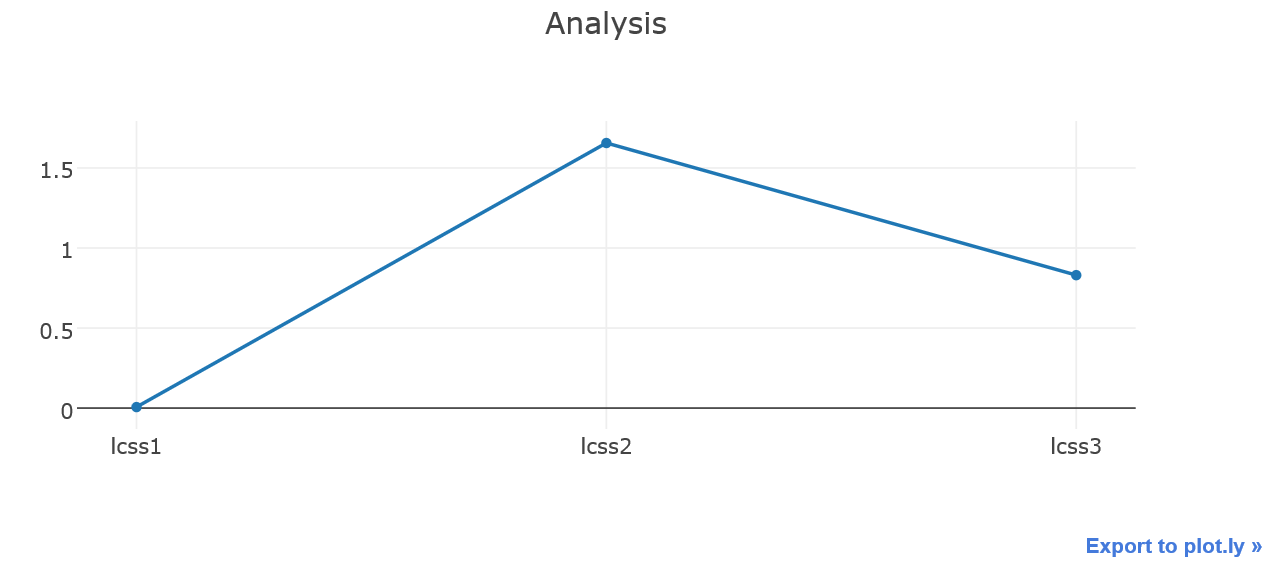
**4. Features:**

* User can upload the desired input string and pattern files.
* User can also run the algorithm for individual Strings of Files.
* The tool provides the following four algorithms and provides a time analysis of the respective algorithms.
  + LCSS
  + KMP
  + Boyer Moore Algorithm
  + Naïve String Search
* We can also do the time analysis of the same input on all the algorithms at once.
* Time analysis is done by generating the time graph of algorithms.

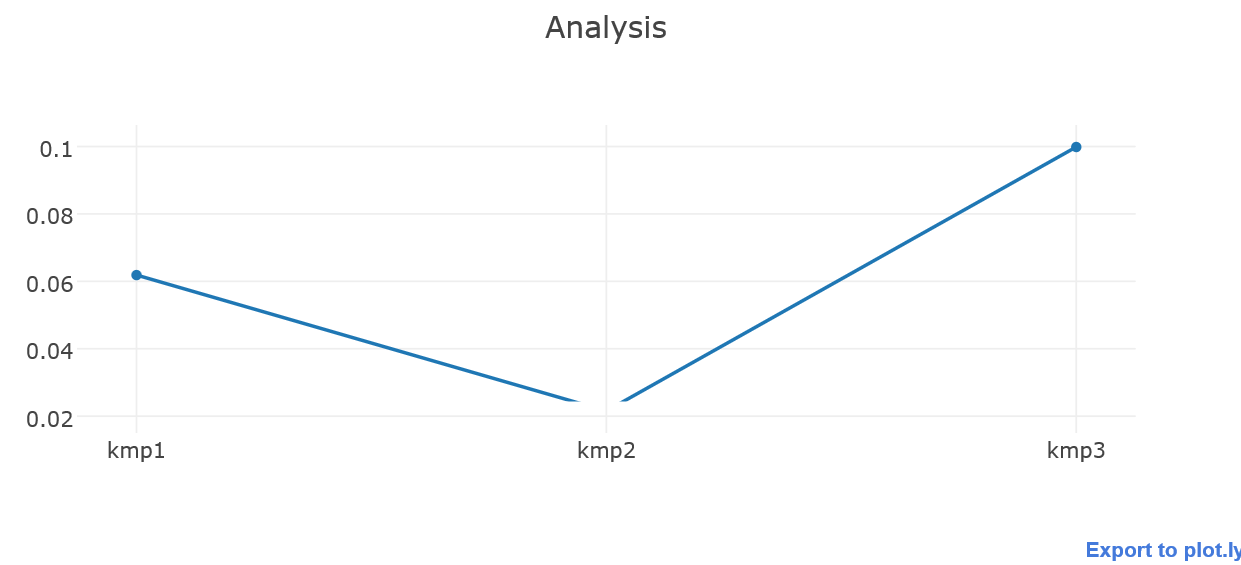
**5. Tool evaluation:**

The table shows the performance analysis of the four algorithms against different dataset sizes:

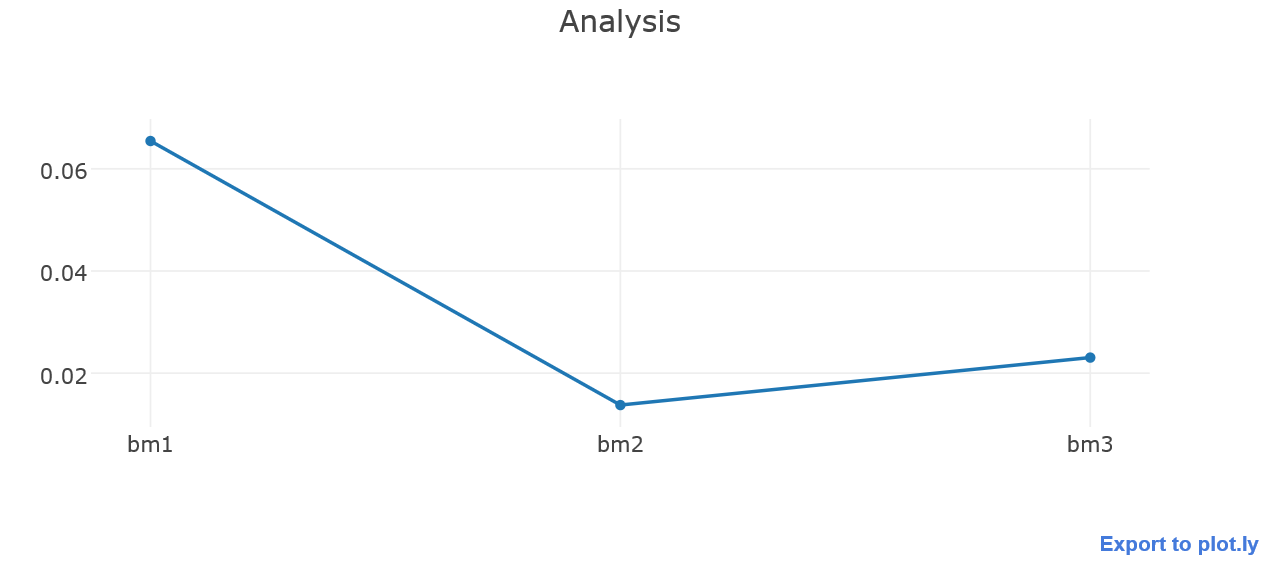
1. **Native string search**
2. **Longest common sub string:**



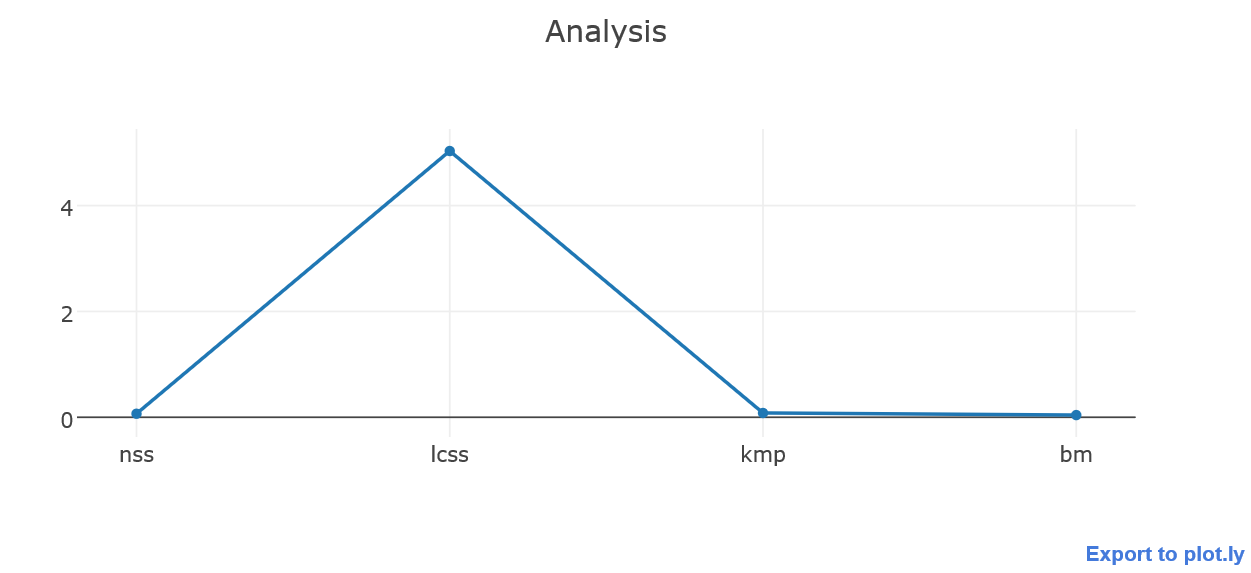
1. **Knuth Morris Pratt Algorithm:**



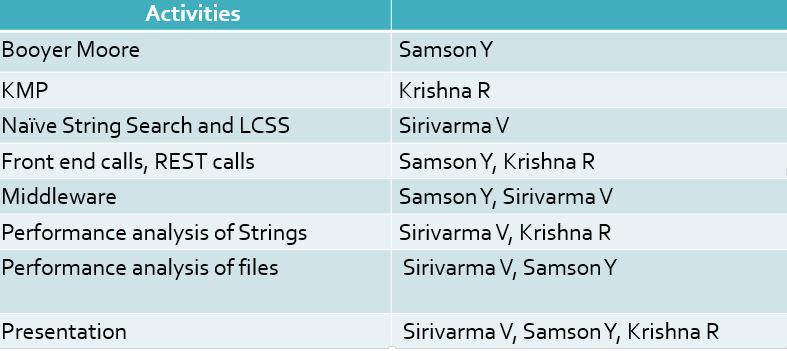
1. **Boyer Moore Algorithm:**



1. **Evaluation of all the four algorithms for three different data set sizes:**



**6. Project Contributions:**

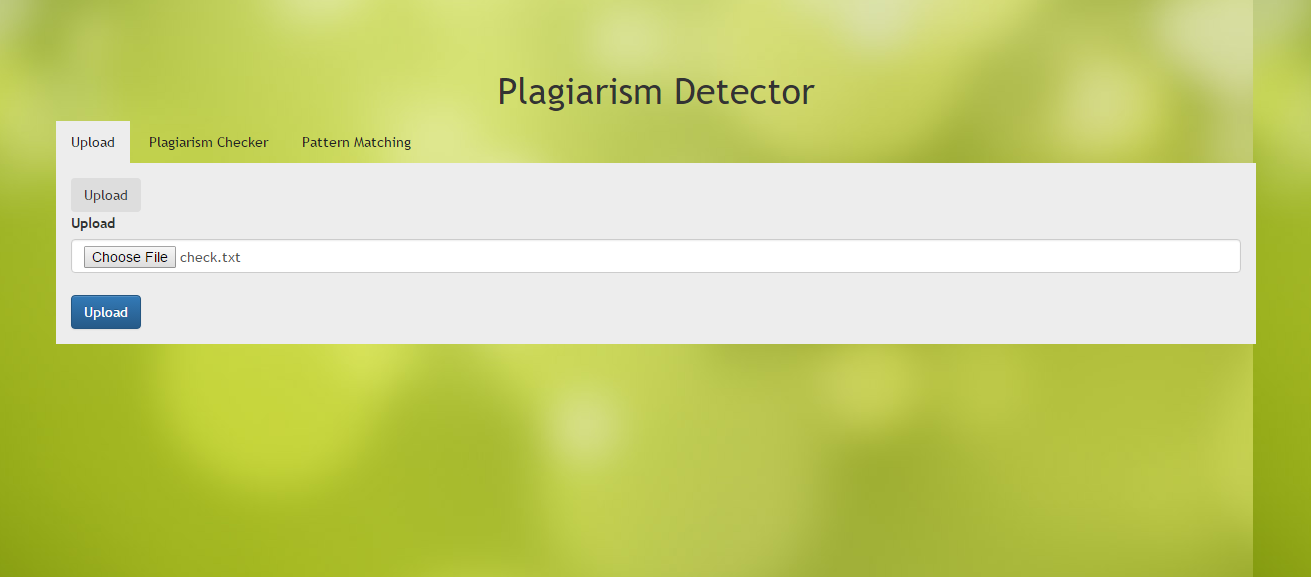


**7. Conclusion:**

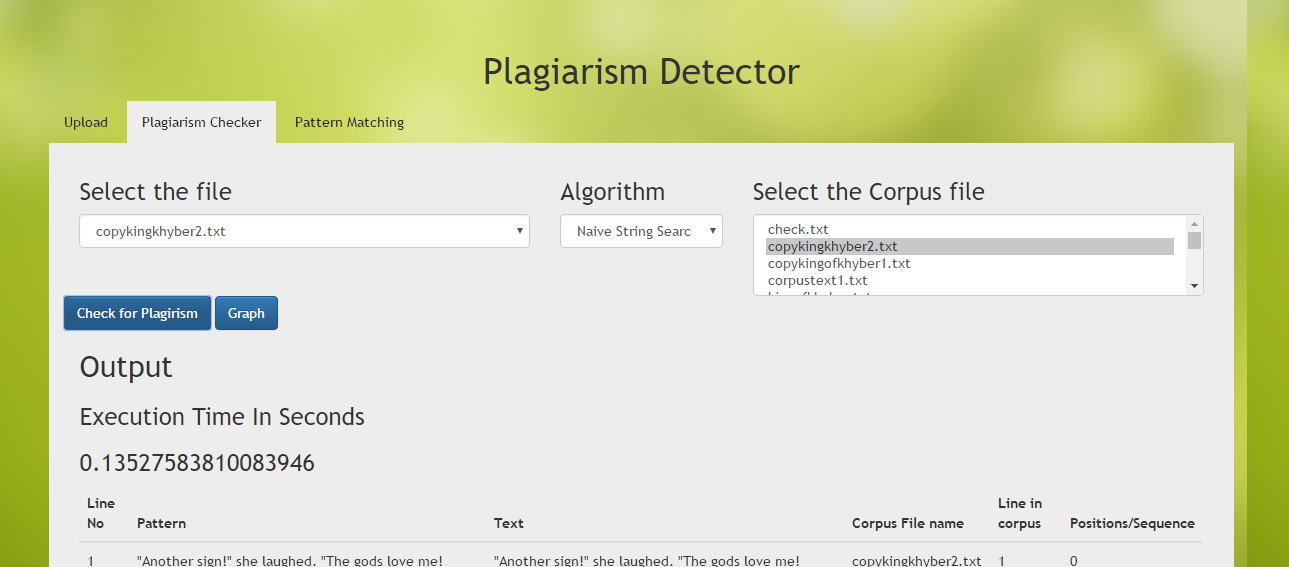
After through time analysis of the time graphs and running time of all the four string matching algorithms on different size datasets we find that Boyer Moore algorithm has the best running time of O(n/m). Secondly, Longest Common Sub Sequence has the worst running time of O(n2).Thirdly, KMP algorithm has better performance han the Naïve String Search algorithm.

**8. Screenshots:**

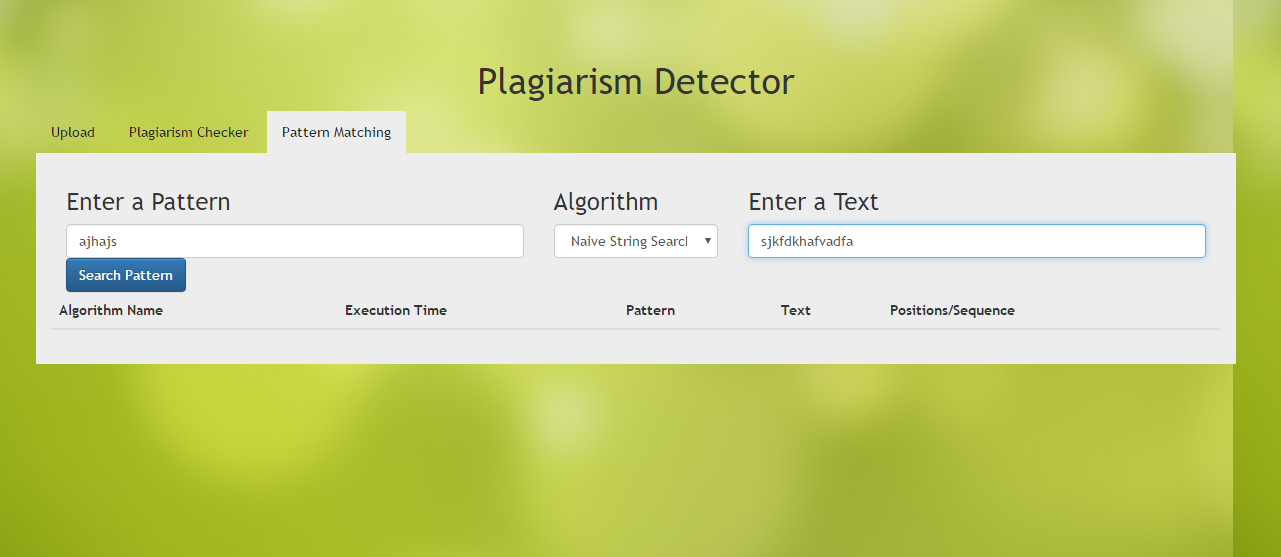
1. **Uploading corpus files to the string matching algorithms:**



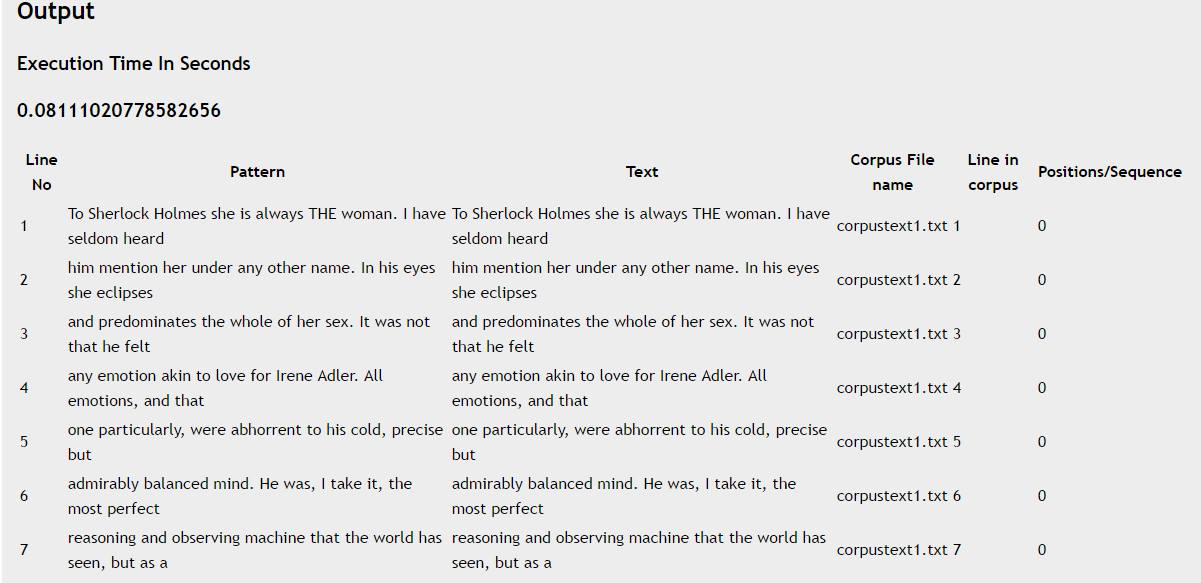
1. **String matching when inputs are text files:**



1. **String matching when input is input string:**



1. **Output showing the matched strings:**



**9. Webbibliography:**

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

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

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

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