SPL-1 Project Report, [2018]

Paper Analyzer

SE-305: Software Project Lab - 1

Submitted by

A.T.M. Fazlay Rabbi

BSSE Roll No. : 926

BSSE Session: 2016-2017

Supervised by

Dr. Kazi Muheymin-Us-Sakib

Designation: Professor

Institute of Information Technology

Institute of Information Technology

University of Dhaka

[30-05-2018]

Table of Contents

1. Introduction

The idea of a paper analyzer tool is to analyze a research paper or find information from the paper in a short time. The target of my project was to make such an analyzer tool. This will identify various information such as Title, Author details, affiliation etc. from the inputted pdf file (basically a conference paper). Furthermore, the relationship between references and the paper is represented graphically.

1.1. Background Study

PDF box:

PDF box is an open source library implemented in java platform to work with PDF documents. This allows user to create new PDF document, manipulation of existing documents and extract contents from the documents. This library is used to parse text from inputted PDF document in this project.

Pattern Matching algorithm:

There are many algorithms to search a pattern in a string such as-
Naïve String Search Algorithm, KMP algorithm, Robin Karp String search algorithm, Boyer Moore string search algorithm, Finding Pattern using Suffix Tree, pattern searching using regular expression etc.
Suffix tree is used to find a string pattern from the text in this project.

Suffix Tree and Ukkonen's algorithm:

A Suffix Tree is the compressed trie containing all the suffixes of the given text. As all the suffixes are in the tree, a pattern can easily be found by matching with any prefix of the suffix.
Ukkonen's algorithm gives the faster and convenient way to implement the suffix tree of the given text. In short, the algorithm is –

- ➔ Make an implicit suffix tree with the first character
- ➔ Step through the string by adding successive characters
- ➔ If any suffix is not found, create a branch from the highest matched substring end character
- ➔ Continue the process
- ➔ Tree completion and pattern searching

IEEE Conference Paper Format:

Sample IEEE Paper for A4 Page Size

First Author[#1], Second Author[*2], Third Author[#3]

[#]*First-Third Department, First-Third University*
Address Including Country Name
[1]first.author@first-third.edu
[3]third.author@first-third.edu

[*]*Second Company*
Address Including Country Name
[2]second.author@second.com

***Abstract*— This document gives formatting instructions for authors preparing papers for publication in the Proceedings of an IEEE conference. The authors must follow the instructions given in the document for the papers to be published. You can use this document as both an instruction set and as a template into which you can type your own text.**

***Keywords*— Put your keywords here, keywords are separated by comma.**

I. Introduction

This document is a template. An electronic copy can be downloaded from the conference website. For questions on paper guidelines, please contact the conference publications committee as indicated on the conference website. Information about final paper submission is available from the conference website.

II. Page Layout

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

A. Page Layout

Your paper must use a page size corresponding to A4 which is 210mm (8.27") wide and 297mm (11.69") long. The margins must be set as follows:

- Top = 19mm (0.75")
- Bottom = 43mm (1.69")
- Left = Right = 14.32mm (0.56")

Your paper must be in two column format with a space of 4.22mm (0.17") between columns.

III. Page Style

All paragraphs must be indented. All paragraphs must be justified, i.e. both left-justified and right-justified.

A. Text Font of Entire Document

The entire document should be in Times New Roman or Times font. Type 3 fonts must not be used. Other font types may be used if needed for special purposes.

Recommended font sizes are shown in Table 1.

B. Title and Author Details

Title must be in 24 pt Regular font. Author name must be in 11 pt Regular font. Author affiliation must be in 10 pt Italic. Email address must be in 9 pt Courier Regular font.

TABLE I
Font Sizes for Papers

Font Size	Appearance (in Time New Roman or Times)		
	Regular	**Bold**	**Italic**
8	table caption (in Small Caps), figure caption, reference item		reference item (partial)
9	author email address (in Courier), cell in a table	abstract body	abstract heading (also in Bold)
10	level-1 heading (in Small Caps), paragraph		level-2 heading, level-3 heading, author affiliation
11	author name		
24	title		

All title and author details must be in single-column format and must be centered.

Every word in a title must be capitalized except for short minor words such as "a", "an", "and", "as", "at", "by", "for", "from", "if", "in", "into", "on", "or", "of", "the", "to", "with".

Author details must not show any professional title (e.g. Managing Director), any academic title (e.g. Dr.) or any membership of any professional organization (e.g. Senior Member IEEE).

To avoid confusion, the family name must be written as the last part of each author name (e.g. John A.K. Smith).

Each affiliation must include, at the very least, the name of the company and the name of the country where the author is based (e.g. Causal Productions Pty Ltd, Australia).

Email address is compulsory for the corresponding author.

C. Section Headings

No more than 3 levels of headings should be used. All headings must be in 10pt font. Every word in a heading must be capitalized except for short minor words as listed in Section III-B.

1) Level-1 Heading: A level-1 heading must be in Small Caps, centered and numbered using uppercase Roman

Figure 1: IEEE Conference Paper Template

JAVA Collection Concept:

To implement the project, I have learned Array List, Hash map.

Array List is used to store data collection. It provides some convenient methods to organize, store and retrieve data without requiring the knowledge of how the data is stored. It also changes size dynamically.

Hash map is a way to map pair of attributes related with each other. A value is mapped with a key so that the value can be retrieved by using the key.

1.2. Challenges

- Constructing Suffix Tree: Difficulties was faced to implement the algorithm for suffix tree construction. By understanding and following the rules mentioned in Ukkonen's algorithm, Suffix tree was successfully made.

- Counting Occurrences of each reference was another challenge. After finding the pattern that how reference number is written in the text the problem was solved. Apart from that finding the reference number in string format and convert it to integer value was also a difficulty.

- The final challenge was to draw a spoke diagram to show the relationship strength between each reference and the paper. As the spoke number of the diagram is dynamic according to total references and thickness of each spoke is changed relating to the occurrences of the references, it was quite a challenge for me to implement that. With the help of geometrical mathematics and understanding graphics implementation of java, the problem was eradicated.

2. Project Overview

To complete this project following tasks were done:

- ➔ At first, all the texts from the inputted pdf document (paper to analyze) were parsed using pdf box.
- ➔ An output.txt file was created to store the parsed text.
- ➔ The expected information was extracted after doing some operations.
- ➔ After that, the output was shown conveniently.

2.1 Implementation:

2.1.1 Design and Diagram

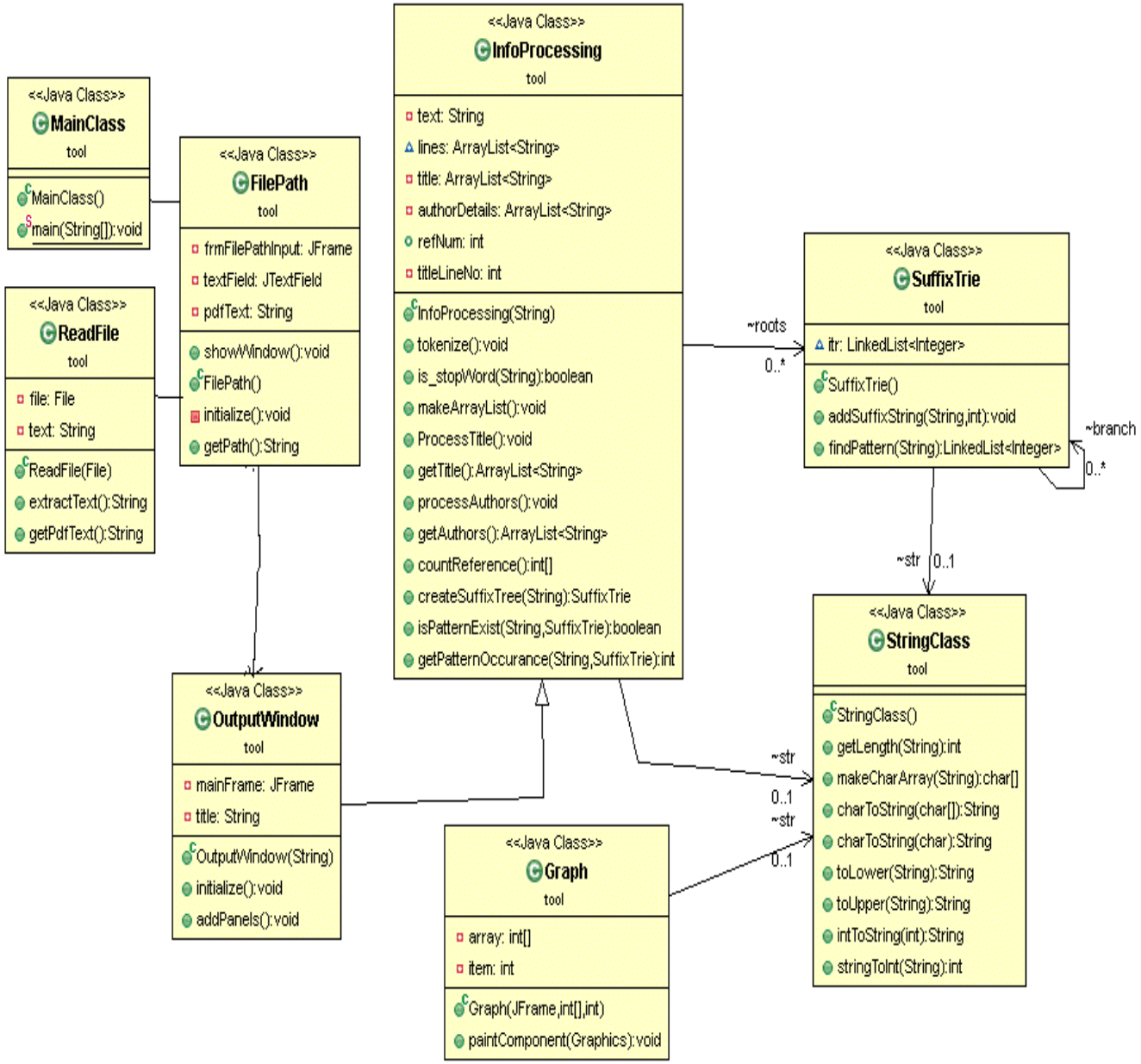

Figure 2: UML Diagram for the project

2.2 Code Analysis:

2.2.1 Package: tool

This package holds the whole implementation of this project.

2.2.1.1 Class: MainClass

main(String[] args)

In here, the main method is called. Program starts executing from here.

2.2.1.2 Class: FilePath

showWindow(): void

This method is to show user Interface window to get the input file or path of the input file and also catches exception if occurs.

FilePath()

This is a constructor for FilePath class.

initialize(): void

This method initialize the main frame for the input window and takes the input.

getPath():String

This method returns the file path string which is inputted.

2.2.1.3 Class: ReadFile

ReadFile(file):

Constructor of the ReadFile class.

extractText(): String

Parses texts from the inputted pdf file.

getPdfText():String

this method returns the extracted texts from pdf.

2.2.1.4 Class: StringClass

StringClass(): constructor of this class.

getLength(String): int

This method takes a string as a parameter and returns the length of that string.

makeCharArray(String):char[]

This method converts a string as a character array and returns it.

charToString(char[]):String

Converts a character array to string and returns it.

charToString(char[]):String

Makes a character to String

toLower(String): String

Converts all the character of a string to lower case and returns the modified string.

toUpper(String): String

Converts all the character of a string to upper case and returns the modified string.

intToString(int): String

Modifies an integer number to string and returns.

stringToInt(String): int

Modifies a string to an integer number and returns.

2.2.1.5 Class: SuffixTrie

SuffixTrie()

Constructor of the class

addSuffixString(String):void

Adds the substring according to the algorithm to the suffix tree.

findPattern(String): LinkedList<Integer>

Takes a pattern as an output and finds in the suffix tree. If match found, returns the index from where the pattern starts.

2.2.1.6 Class: InfoProcessing

InfoProcessing(String)

Constructor of this class.

makeArrayList():void

makes list of each line of the text document parsed from pdf

processTitle(): void

extracts title from the text.

getTitle():ArrayList<String>

returns title of the text as an arrayList of each line.

processAuthors():ArrayList<String>

returns author details of the paper as an arrayList of each line.

isPatternExist(String,SuffixTrie): Boolean

Checks if a pattern exists in a string or not

getPatternOccurence(String, suffixTrie): int

If a pattern exists, then this class will return the number of occurrences of that pattern.

2.2.1.7 Class: Graph

paintComponent(Graphics):void

Overrides the method of Graphics class and paints the spoke diagram as needed.

2.2.1.8 Class: OutputWindow

initialize(): void

initializes the output window to be appeared.

addPanels():void

In this class frames, panels and other components are created. Different components are added to panel. Panels are added to frame.

3. User Manual

3.1 Requirements

The project is platform independent.

The project was developed in JAVA. So JRE (Java Runtime Environment) must be installed.

3.2 PDF Box Library

This open source library should be integrated with this project. In this project, version 2.0.8 was used.

3.3 Taking Input

If the project runs successfully with all requirements fulfilled, then the following window will appear to take input. User should give a valid input.

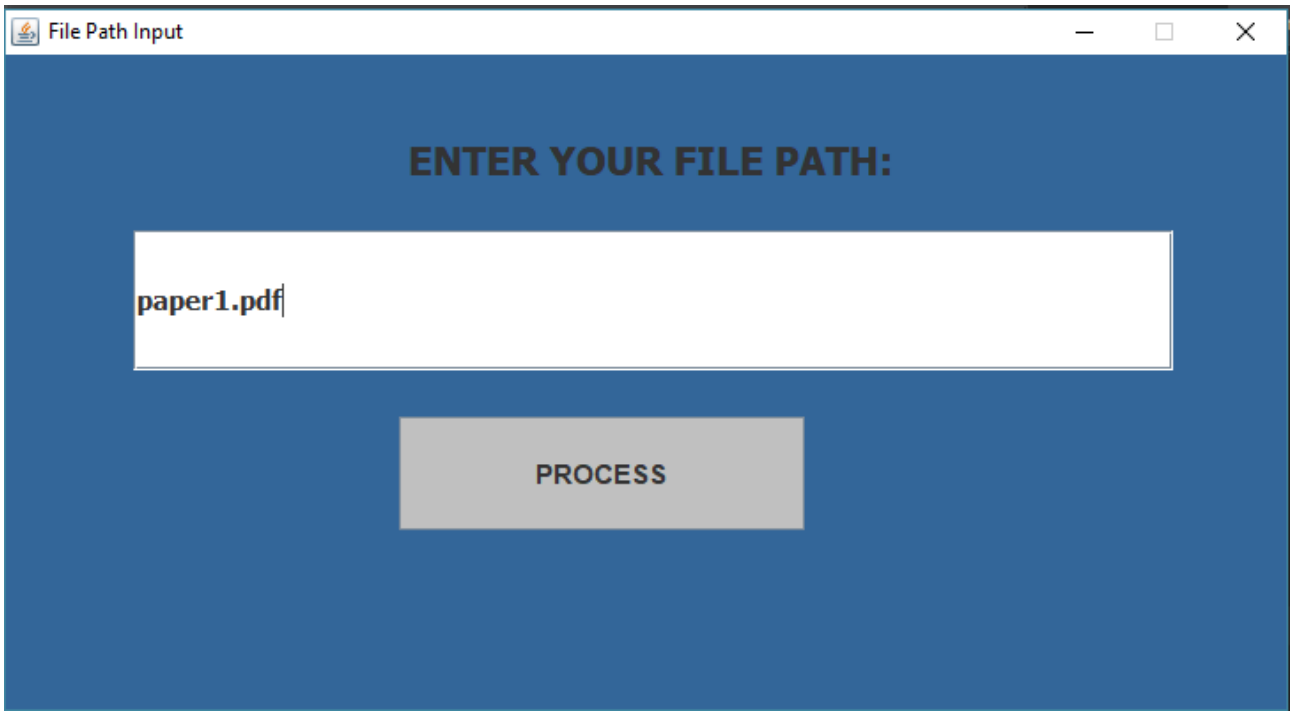

Figure 3: Input Window

3.4 Output

After clicking the button “PROCESS” following output window appears:

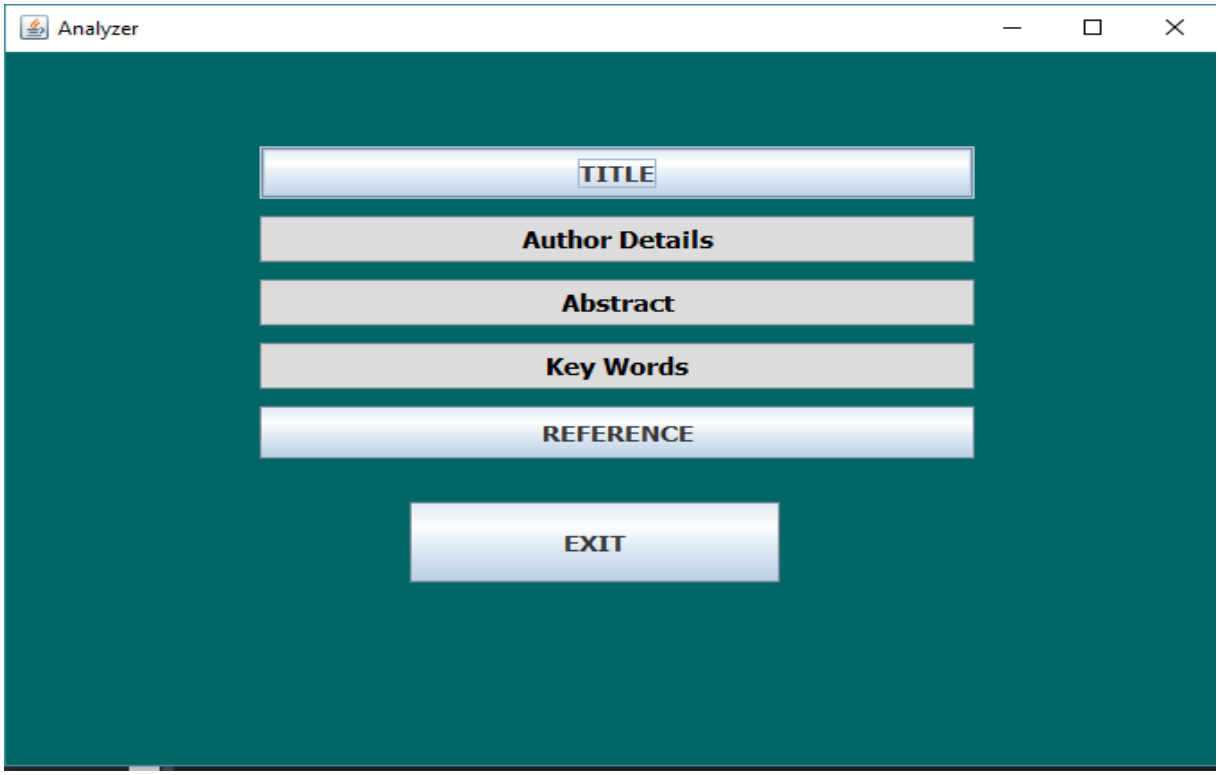

Figure 4: Analyzer window

Sample output for “TITLE”

Figure 5: Title Output Window

Sample REFERENCE output

Shows relationship strength between paper and references. The thicker is the line, the stronger is the relationship.

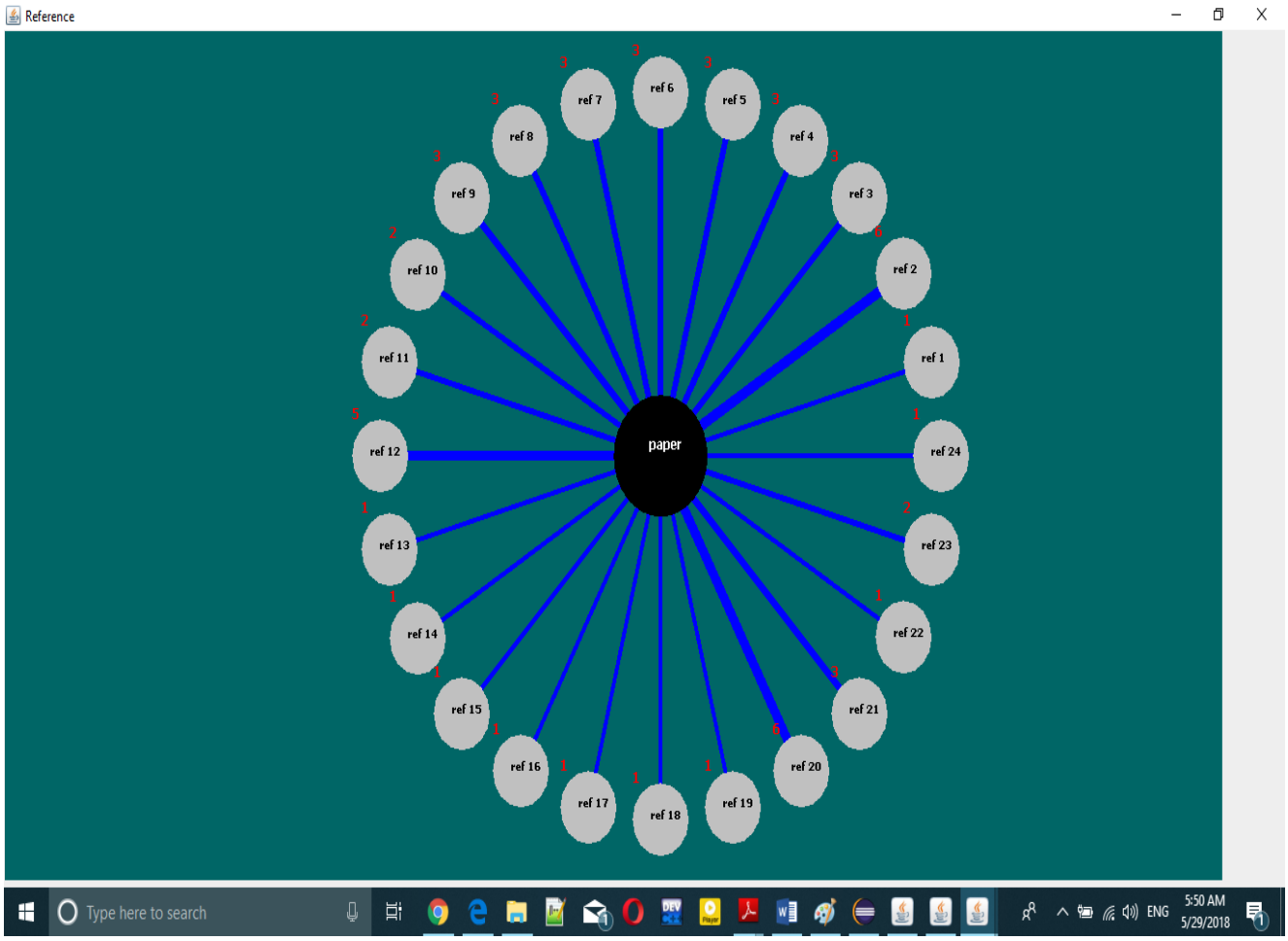

Figure 6: Sample Reference Output Window

4. Conclusion

I have gained so much experience during this project. This project has taught me how to face challenges and overcome them. I have learned to design implementation and many more topics related to this project. Also, during this project lifetime, my coding skill has certainly increased. I am very grateful to my supervisor for his guidance. Hopefully, the learnings from this project will help me out to do better in the future projects.

References

[1] https://www.javatpoint.com/java-arraylist, Java ArrayList Class, last accessed on last accessed on 10 FEB 2018

[2] https://www.geeksforgeeks.org/ukkonens-suffix-tree-construction-part-1, Ukkonen's Suffix Tree Construction – Part 1, last accessed on last accessed on 16 MAR 2018

[3] https://docstore.mik.ua/orelly/java-ent/jfc/ch04_05.htm, Graphics with AWT and JAVA 2D, last accessed on last accessed on 29 APR 2018

[4] JAVA How to Program, Paul Deitel and Harvey Deitel, Tenth Edition, Page no. 287,288,561.

My **GITHUB** link of this project:
https://github.com/frabbi1/SPL1/tree/master/Final_Implementation