Software Requirements Specification

Of

IMERS

Deadliners

Fatih Burak Belce

Mustafa Güven

Oğuzhan Demir

Özgür Baskın

Contents

1. Introduction

This software requirement specification (SRS) report has deep and branched description for our project Index Mechanism for Regex Search which stands for the cases where a regular expression is used to match a string pattern from a large set of documents or crawled web pages. The project is planned to be implemented by a group of senior students, which is called Deadliners, from Department of Computer Engineering in Middle East Technical University.

1.1 Problem Definition

Regular expressions are widely used in a variety of tasks such as text processing, textual data mining, data validation, data scraping and simple parsing. Although there are different implementations of Regular Expression libraries and applications within different layers such as command line tools (like grep or sed), scripting languages (like Perl or Python), common programming languages (java, c++ etc.), few implementations (MySQL, PostgreSQL, Oracle) exist in data tier (i.e.. database layer).

However, in the software product environment, none of the implementations above take advantage of an index structure, which results in searching and matching operations to be executed over all input string sequences without shrinking down the number of searched documents, resulting a huge look up scope for a simple regex query. This project aims to develop a regular expression engine which uses a specialized (k-gram) index structure to speed up operations over large number of documents by decreasing the amount of them. A short research would reveal that similar studies which use indexes exist, but none of the solutions utilize such kind of specialized index for regular expression pattern matching queries.

1.2 System Overview

System has three major parts. Query parser, plan generator and execution engine.

Query parser has two components; Regex parser and regex normalizer.

Regex parser is the first component and it's the first one that interacts with the user. User, at this phase, could specify the documents that will be used by the engine, or the path which the engine would start the match operation to start. Then the duty of regex parser starts. It determines all type of symbols that a regex can have, pushes them to a stack to create a form so that the component coming after regex parser can use the parsed version of the query.

Regex normalizer takes the query created by the regex parser, it makes some operations and another level of string query is created at the end. It can be said that

after this operation, string query only includes specific types of symbols, other symbols are converted to this specific types. So that second major component of our engine can take the query string and process it. The name “normalizer” comes from this query string simplification.

Plan generator also has two parts; One being the logical plan generator, other being the physical plan generator.

Logical plan generator literally generates a plan for the first phase of index construction. Our specialized index has terms called “grams”. After this logical plan, a tree holds all the necessary paths to our grams, but without excluding the useless grams. Those grams are turned up to be useless by selectivity equation. These terms are explained and well defined in the part 1.3 Definitions, Acronyms and Abbreviations.

Physical plan generator generates a plan for the second phase of index construction. Since constructing an index is a big concern in our project, index construction takes several steps. At this phase, the tree structure coming from the logical plan generator is used and processed. Only the useful grams need to be at the leaf nodes at the end of the process. Almost all of the nodes and branches are altered, some of them are gone and some new branches are added. There may be no useful gram left, which would give unsuccessful index creation. To prevent this, some precautions are taken in this physical plan generator phase.

Execution Engine has two components as well; Index look up and in memory matching.

The design of the index, the entries to look up and the index look-up order (execution plan) have a direct and very significant impact on performance. Index look up is very fast since all the keys in the index can be cached in main memory due to its small size. Last query string coming from the physical generation plan traverses in the index, matches some of the documents, and just the documents that have the grams are returned. On the other hand, page to page reading is done to get the pages to memory as a preparation for the next phase.

In memory matching is the last phase in the regex machine. Returned documents are brought to memory and a matching engine matches the strings by comparing them to show them to the user. At the end, depending on the user's preferences, whole page that contains the matching pattern, or the terms can be delivered.

1.3 Definitions, Acronyms, and Abbreviations

Class Diagram	In the UML, it is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
CLT	Command Line Tool
Debian	It is a Unix-line computer operating system that is composed entirely of free software, most of which is under the GNU General Public License, and packaged by a group of individuals known as the Debian project.
DFA	Deterministic finite automaton
IEEE	IEEE Institute of Electrical and Electronics Engineering
Imers	Indexing Mechanism for RegEx Search
GNU	It is a recursive acronym for "GNU is not Unix!", and an operating system whose design is Unix-like, but differs from Unix by being free software and containing no Unix code.
Grep	It is a command-line utility for searching plain-text data sets for lines matching a regular expression.
GUI	Graphical User Interface
Lucene	It is a free and open source information retrieval software library, originally written in Java.
Parser	It is the component that makes the process of analysing a string of symbols, either in natural language or in computer

	languages, conforming to the rules of formal grammar.
POSIX	It defines the application programming interface, along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems.
Regex	Regular Expression
Sed	It is a Unix utility that parses and transforms text, using a simple, compact programming language.
SRS	Software Requirements Specification
Throughput	The required volume of data transfer per unit time.
UC	Use Case
UI	User Interface
UML	Unified Modeling Language
Unix	It is a family of multitasking, multiuser computer operating system developed in 1970's.
User	The human being interacting with the system

1.4 Assumptions and Dependencies

IMERS do not require an internet connection. When compiled from the source code, it can be used in any platform. But this case would need a C++ compiler and some additional libraries. This is explained in detail at section 2.3 Constraints. Though, finished and compiled product can be available in UNIX, Windows or Mac platforms to reach the users directly. In this case, all the user has to do is to click the application icon and choose the options in there, or open up a terminal and use IMERS keyword and the parameters desired to make the search more detailed, then

hit enter. Also depending on the users' choice, certain documents or web pages are searched.

2. Overall Description

Main goal of this section is to give the general factors affecting the system and its requirements. There are 3 sub-sections which are product functions, Interfaces and Constraints for describing the system as a whole.

2.1 Product Functions

This system is designed to serve for one type of user only who wants to find a string pattern among the documents he decided to search for. There is no administration, no login required to use the system. That's why there will be only a single actor in all of the use cases. Two major use cases will be explained in detail. This is because there would be two different interfaces that require different functionality and different input expectations from the user. One may need a button click action; the other may need an extra command line argument. This situation leads to different use case scenarios. Same functions will be used after some point though (after user provides the input and selects the options, the string query will follow the same route until the search results are delivered), but when the user interaction part comes into action, there will be several different option selecting features that may be utilized. This will be clearer after the explanation of the use cases.

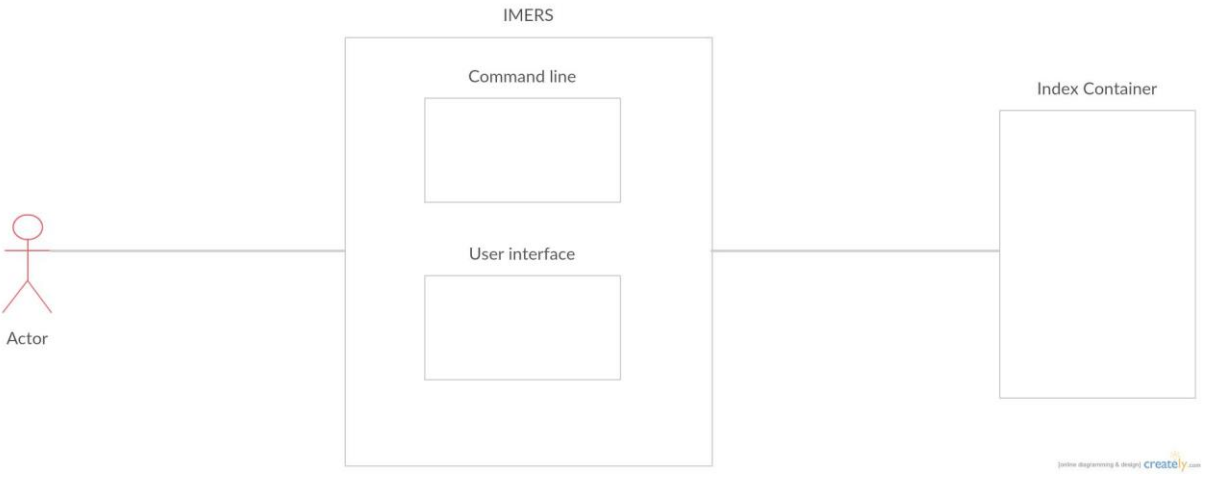

Figure 1 - Block Diagram

2.1.1 Use-Case Model Survey

2.1.1.1 Use-Case for CLT

Figure 2 – Use Case Diagram for CLT

2.1.1.1.1 Query Regex Input

Users type the query string into command line.

Use Case Name	Query Regex Input
Use Case ID	UC1
Primary Actor	User
Precondition	User should type "imers" first.
Trigger	User types the string and presses "Enter"
Main Success Scenario	1. User opens the command line 2. User types "imers" and the string 3. System returns the paths of the files having matches.
Alternate Scenarios	--

Use Case Scenario 1: Query Regex Input

2.1.1.1.2 Choose Option

Users type the query string with options (flags) to specify the output. If a flag is not set by the user, default search will be the same with Query Regex Input which is mentioned in 2.1.1.1.1.

Use Case Name	Choose Option
Use Case ID	UC2
Primary Actor	User
Precondition	User should type "imers" first
Trigger	User types flag(s) and the string after a space character, then presses "Enter".
Main Success Scenario	1. User opens the command line. 2. User types "imers" and the string. 3. System returns paths of the files having matches.

Alternate Scenarios 1	1. User opens the command line. 2a. User types "imers" and enters –f with a filename in the current directory or a path to the file. One space is left between each. 2b. User types "imers" and enters –d with a directory under the current directory or a path to the directory. One space is left between each. 3. User types the string. 4a. System returns the lines having matched strings in the specified file for –f flag. 4b. System returns paths of the files in the specified directory for –d flag.
Alternate Scenarios 2	1. User opens the command line. 2a. User types "imers" and enters –i after a space character. 2b. User types "imers" and enters –v after a space character. 3. User types the string. 4a. System ignores case sensitivity and returns paths of the files for –i flag. 4b. System inverts the sense of matching, to select non-matching lines and then returns paths of the files for –v flag.
Alternate Scenarios 3	1. User opens the command line. 2a. User types "imers" and enters –c after a space character. 2b. User types "imers" and enters –o after a space character. 3. User types the string. 4a. System counts matches and returns number of the matches for –c flag. 4b. System returns only matching parts of the lines for –o flag.
Alternate Scenarios 4	All flags can be used together. 1. User opens the command line. 2. User types "imers" and leaves a space character.

	3. User may enter the flags in two ways: After a "-" character, all flags can be typed without spaces such as –ic. OR, all flags follow each other such as –i –c. 4. User types the string after a space character.
Exception Scenario	If –E flag is not provided in Alternate Scenarios 4, system gives an error.

Use Case Scenario 2: Choose Option

2.1.1.1.3 Specify Regex Type

User chooses the regex type whether it is basic regex or extended regex. The default case is basic regex. In order to choose extended regex, -E flag is typed by the user.

Use Case Name	Specify Regex Type
Use Case ID	UC3
Primary Actor	User
Precondition	User should type "imers" first
Trigger	User types the string and presses "enter". The flag is typed between "imers" and the string if provided.
Main Success Scenario	1. User opens the command line. 2. User types "imers" and the string. 3. System returns paths of the files having matches.
Alternate Scenarios	2a. If –E flag is provided, system treats regex operands as normal characters and returns the output accordingly. For example, "imers -E '{1'" searches for the two-character string "{1".
Exception Scenario	If –E flag is not provided in the above example, system gives an error.

Use Case Scenario 3: Specify Regex Type

2.1.1.1.4 Use Manual

User types "man imers" or "imers -h" to show manual page and get information about usage.

Use Case Name	Use Manual
Use Case ID	UC4
Primary Actor	User
Precondition	--
Trigger	User types "man imers" or "imers -h" and presses "enter".
Main Success Scenario	1. User opens the command line. 2. User types "man imers" or "imers -h". 3. System shows the manual page of imers.
Alternate Scenarios	--

Use Case Scenario 4: Use Manual

2.1.1.2 Use-Case for UI

Figure 3: Use Case Diagram for UI

2.1.1.2.1 Query Regex Input

Users type the query string into search box.

Use Case Name	Query Regex Input
Use Case ID	UC5
Primary Actor	User
Precondition	-
Trigger	User types the string and presses "Enter" or "Search" button
Main Success Scenario	1. User opens the UI 2. User types the string using keyboard 3. System returns the paths of the files and/or content of the file having matches.
Alternate Scenarios	1. User opens the UI 2. User types the string using on-screen keyboard 3. System returns the paths of the files and/or content of the file having matches.

Use Case Scenario 5: Query Regex Input

2.1.1.2.2 Select File or Directory

Users choose the file or the directory that will be searched by clicking files button.

Use Case Name	Select file or directory
Use Case ID	UC6
Primary Actor	User
Precondition	-

Trigger	User clicks the files button
Main Success Scenario	1. User opens the UI 2. User clicks the files button 3. System returns the paths of the files and/or content of the file having matches.
Alternate Scenarios	-

Use Case Scenario 6: Select File or Directory

2.1.1.2.3 Control Matching

Users choose matching type by clicking control button. Matching options that can be chosen are case sensitivity and inverse matching.

Use Case Name	Control matching
Use Case ID	UC7
Primary Actor	User
Precondition	-
Trigger	User clicks the control button
Main Success Scenario	1. User opens the UI 2. User clicks the control button 3. User chooses case sensitivity matching 4. User types the string. 5. System returns the path or file accordingly
Alternate Scenarios	1. User opens the UI 2. User clicks the control button 3. User chooses inverse matching 4. User types the string. 5. System returns the path or file accordingly

Use Case Scenario 7: Control Matching

2.1.1.2.4 Display Search History

User can display the search history by pressing history button.

Use Case Name	Display search history
Use Case ID	UC8
Primary Actor	User
Precondition	-
Trigger	User clicks the history button
Main Success Scenario	1. User opens the UI 2. User clicks the history button 3. System displays the search history
Alternate Scenarios	-

Use Case Scenario 8: Display Search History

2.1.1.2.5 Specify Regex Type

Users specify regex type by clicking split button. By default regex type is basic.

Use Case Name	Specify regex type
Use Case ID	UC9
Primary Actor	User
Precondition	-
Trigger	User clicks the files button
Main Success Scenario	1. User opens the UI 2. User clicks the Split button 3. By default Basic regex is chosen 4. System returns the paths of the files and/or content of the file having matches with basic regex.

Alternate Scenarios	1. User opens the UI 2. User clicks the Split button 3. User chooses Extended regex 4. System returns the paths of the files and/or content of the file having matches with extended regex.

Use Case Scenario 9: Specify Regex Type

2.1.1.2.6 Control Output Format

Users control the output format by clicking format output button. Users can specify number of matches, only matching parts, font, color of found result, etc.

Use Case Name	Control output format for UI
Use Case ID	UC10
Primary Actor	User
Precondition	-
Trigger	User clicks the Output format button
Main Success Scenario	1. User opens the UI 2. User clicks the Output format button 3. User clicks count button 4. System returns the result and word count
Alternate Scenarios 1	1. User opens the UI 2. User clicks the Output format button 3. User clicks matching parts button 4. System returns only the matching parts instead of whole text.
Alternate Scenarios 2	1. User opens the UI 2. User clicks the Output format button 3. User clicks color pallet and chooses a color to specify color of the matching string

	4. System returns the result as desired
Alternate Scenarios 3	1. User opens the UI 2. User clicks the Output format button 3. User clicks font button to specify font of the output 4. System returns the result as desired

Use Case Scenario 10: Control Output Format

2.1.1.2.7 Control Output File

Users choose output file by using pager buttons. The number of pager buttons is equal to the files that contain the resulting string.

Use Case Name	Control output file
Use Case ID	UC11
Primary Actor	User
Precondition	-
Trigger	User clicks one of the pager buttons
Main Success Scenario	1. User opens the UI 2. User clicks one of the pager buttons 3. User chooses a file among all results 4. System shows the desired file
Alternate Scenarios	-

Use Case Scenario 11: Control Output File

2.1.1.2.8 Create Log

User create log file in txt format by clicking log button.

Use Case Name	Create log
Use Case ID	UC12
Primary Actor	User
Precondition	-
Trigger	User clicks the log button
Main Success Scenario	1. User opens the UI 2. User clicks the log button 3. System saves the result into a txt file
Alternate Scenarios	-

Use Case Scenario 12: Create Log

2.1.1.2.9 Use Manual

User clicks help button to show manual page and get information about usage.

Use Case Name	Use manual for UI
Use Case ID	UC13
Primary Actor	User
Precondition	-
Trigger	User clicks the help button
Main Success Scenario	1. User opens the UI 2. User clicks the help button 3. System shows the manual page of imers.
Alternate Scenarios	-

Use Case Scenario 13: Use Manual

2.2 Interfaces

2.2.1 User Interfaces

There will be two kinds of interfaces in the system. One being the command line tool, the other is the GUI. That's been said; command line tool has pretty much nothing but a simple text. User shall type the necessary commands in order, and shall be able to get the results for the pattern he looks for as a simple text printed on the command line. There will be cases that only paths are printed or the lines are printed which the pattern matches. There is no necessity of adding the screen captures of terminal since its flow will be exactly like it is explained in the section 2.1.1. Though, for the GUI part it is needed to put the screenshots of the system, since it is not clear where to click or where to select to reach to the next step. There will be two major pages in the GUI; the first is the search page, the second is the result page.

2.2.1.1 Search Page

There is a search button just below the search box, which is supposed to start the search process when clicked, or when the user hits enter on the keyboard. Search box is obviously there to write the regex string. Just right on the search box there is symbol which displays a small keyboard. It eases the pattern writing process for users since it only has the necessary regex symbols and characters. At the top left, there is a file button. This one, when clicked, opens up a window for user to go through the directories it has on its computer and to specify the path or files that user wants them to be searched. Right next to it there is the settings-like symbol, we call it control button, which displays the options of the search. All of the available options are explained in the section 2.1.1.2 use cases for the UI. The one on the right next to the control button is the history button. User utilizes this one to look for his previous searches. On the top there is a split button for the two regex options available. User can click on this and choose between basic or extended regexes.

In the middle top, a path can be seen. It is where the user is searching. It changes when user changes its directory. At the right of that there is a control output format button. This one is to alter the type of output; a lot of options are available as explained in the use-case section. Next to this on the right, there is a pager button. User here chooses a file among other files that have the "matched regex pattern". At the top right, there is a log button. This one is to save the search results into a txt file.

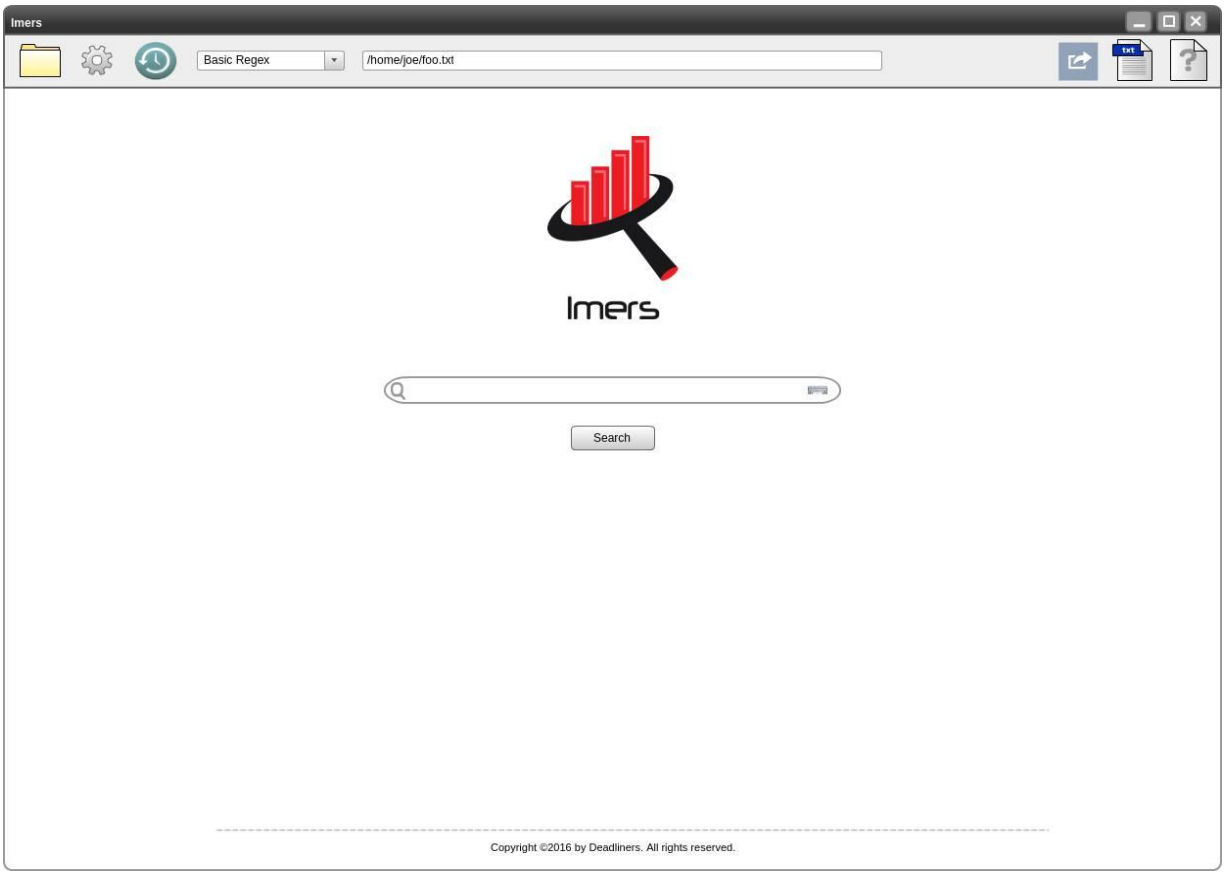

Figure 4: GUI for Main Page Imers

2.2.1.2 Result Page

In this page resulting files and their content are shown. Top elements of the page remain the same. However; the search box, search button and logo are moved to the top and they are replaced with the resulting text at the middle. The bottom of the page there are paging numbers. Contrary to the common search engines, these paging numbers change the file other than going to the next set of results at the same file. Next set of results in the same file are shown using the scroll on the right of the results text. Matched patterns in the text are highlighted using a colour; this option is subject to change, by using the control format button as stated at the previous section, but not necessarily. It depends on the user's choice.

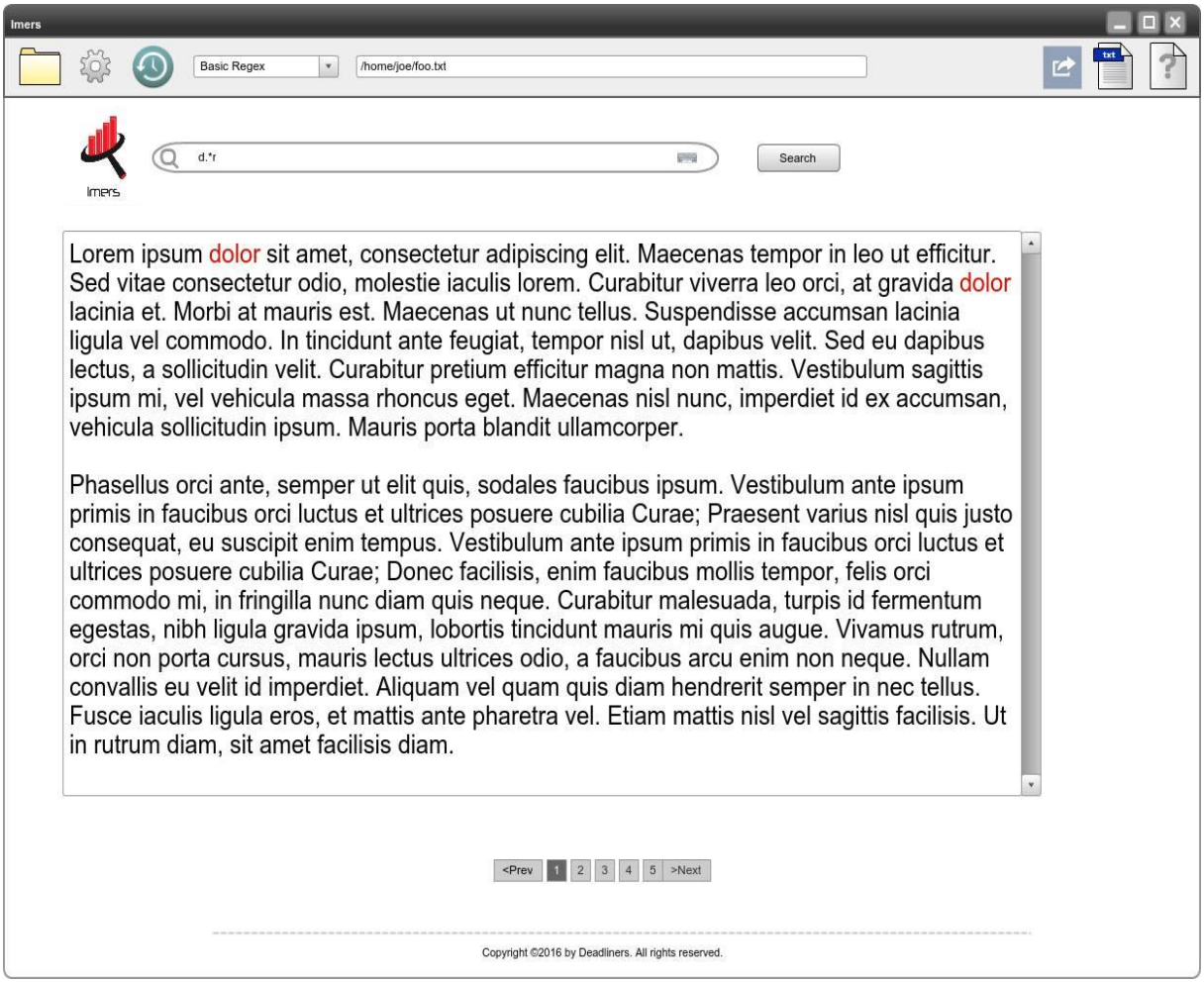

Figure 5: GUI for Result Page Imers

2.2.2 Hardware Interfaces

There are no specific hardware requirements for this project. This project is expected to work on any environment regardless of the hardware specifications.

2.2.3 Software Interfaces

System is implemented on both Windows 10 and Ubuntu 14.04 operating systems. G++ compiler is used along with the flag C++11 to make use of C++11 features. Index is intended to be loaded in to main memory; no database access is made during the implementation. Since this project doesn't require a user to login, or doesn't need to hold information or state of any object, it doesn't make use of a database. A table of software tools used while implementing the project is put below.

Name	Version
Linux Ubuntu	14.04
G++	4.8.4
Lucene	5.2.1

2.2.4 Communication Interfaces

The system won't have any communication through internet or don't have any access to database to get information, it can be used offline and doesn't require a login since it searches a string pattern among the documents.

2.3 Constraints

- The system shall be available anytime possible.
- The system shall be implemented in C++11
- The system shall be tested using a benchmark. Benchmark shall be implemented in Python.
- A visual demo of the project shall be implemented in JFrame.
- There is no constraint about the length of the string.
- There is no constraint about the size of the documents that shall be searched.
- The results shall be displayed depending on the user's choice. It can return the pages, lines or it can give the direct matching string.

3. Specific Requirements

This section involves the subsections inclusively describing functional and nonfunctional requirements and their displays in UML diagrams.

3.1 Functional Requirements

The fundamental functional requirements of the system depicted in use case diagram in Figure 1 and more are described in this section. In addition to these, there exists sequence diagrams which show the flow of actions occurring in a use case for each individual feature.

3.1.1 Functional Requirements for CLT

3.1.1.1 Query Regex Input

- The user of the application shall open the terminal and write the IMERS command first

- User shall write regex query string and shall hit enter.
- System shall show the results.

3.1.1.2 Choose Option

- User shall choose files, or a directory or shall specify basically a path to that directory or file
- User shall control the matching pattern by adding appropriate flags. Depending on the type of flag used, this shall use the pattern as it is OR it shall ignore case distinctions, so that the regex matcher machine would decide whether to ignore the upper and lower case characters or not OR it shall return the non-matching lines instead of matched ones.
- User shall control the return type of output. Depending on the flag type used, this shall return count of matched lines OR it shall return the count of non-matching lines OR it shall only return the matched parts of matching lines.
- Select match type use case is both extends the "choose option" and "specify regex type" use cases; therefore explained in section 3.1.3 Specify Regex Type.

3.1.1.3 Specify Regex Type

In basic regular expressions the meta-characters '?', '+', '{', '|', '(', and ')' lose their special meaning; instead the backslashed versions '\?', '\+', '\{', '\|', '\(', and '\)' are used and the meta-characters are treated as strings. This means that normally '{1' in a regex query would give a syntax error, but extended expressions would treat this as a string and no error is given.

- User shall specify if the basic or extended version of regex control mechanism is used by the regex parser engine.

3.1.1.4 Use Manual

- User shall type "man imers" or "imers -h" (h as an abbreviation for help) in CLT to show the manual and list the options that are available for the use of regex machine.

3.1.2 Functional Requirements for UI

3.1.2.1 Query Regex Input

- User shall open the UI and type the string first.
- System shall show the results which are paths of matching files and/or contents of the files.

3.1.2.2 Select File or Directory

- User shall open the UI.
- User shall choose the file by clicking the files button to specify the search.

- System shall show the results which are paths of matching files and/or contents of the files.

3.1.2.3 Control Matching

- User shall open the UI.
- User shall click the control button and choose a matching type from the list. The list includes case sensitivity and inverse matching. Depending on the selection, this shall ignore case distinctions OR it shall choose non-matching lines, respectively.
- System shall show the results which are paths of matching files and/or contents of the files accordingly.

3.1.2.4 Display Search History

- User shall open the UI.
- User shall click the history button to see the search history.
- System shall display the search history.

3.1.2.5 Specify Regex Type

- User shall open the UI.
- User shall click the split button to choose regex type. There are two regex types; by default basic regex is chosen. Depending on the selection, this shall consider the matchings with basic regex OR it shall consider the matching with extended regex.
- System shall show the results which are paths of matching files and/or contents of the files accordingly.

3.1.2.6 Control Output Format

- User shall open the UI and type the string
- User shall click the output button to choose one of the options which are count, matching parts, color pallet and font. Depending on the selection, this shall count the number of matches OR it shall consider only matching parts which are the lines of the related file OR it shall specify the color of the matching string OR it shall specify the font of the matching string, respectively.
- System shall show the result as desired.

3.1.2.7 Control Output File

- User shall open the UI and types the string.
- User shall click one of the pager buttons which are the files matched with the query string. User shall choose a file among them.
- System shall show the desired file.

3.1.2.8 Create Log

- User shall open the UI and click the log button.
- System shall save the log and write it into a txt file.

3.1.2.9 Use Manual

- User shall open the UI and click the help button.
- System shall show the manual page of imers.

3.1.3 External Interface

Interfaces are explained in the section 2.2. Also they are discussed in detail using figures as well. There are no extra requirements that can be explained in this section. It would be a repetition to mention them again in here.

3.2 Nonfunctional Requirements

The substantial requirements that can be used to judge the operation of a system, rather than specific behaviours, namely, usability, reliability, performance and, supportability are explained in this section.

3.2.1 Usability

This section of non-functional requirements describes the level of ease of use and is examined in two subsections; namely, user-friendliness and accessibility.

3.2.1.1 User-Friendliness

- When the ease of use is considered, this project can be usable as any command line program. Almost all the command line programs have options part to introduce some deep and detailed functionality. This project has that as well. After seeing a simple example, an average user should be able to use the tool directly for its needs.
- Though, to become a guru of this software, one should be able to explore all of the functionalities, which requires a complete reading of the manual and experiencing all the features that the project provides. This even wouldn't last more than a single day.
- While implementing this project, producers of the project are trying their best to ease the job of the users, taking the most user-friendly search applications on the market as examples.

3.2.1.2 Accessibility

- Any user shall be able to access the product any time he/she wants.

3.2.2 Reliability

This section of non-functional requirements describes the level of fault-tolerance required by the system. What makes the system reliable are examined in following five subsections; namely, accuracy, precision, availability, redundancy and error-handling.

3.2.2.1 Accuracy

- The results of any regex search are intended to be as accurate as it can be.
- Scoring and keeping statistics about previous searches are thought to be used for more accurate results, when there is an enormous number of documents over which the search is carried out.

3.2.2.2 Precision

- Any regular expression which is to be used for either basic search or extended search must be resolved in a precise way such that, no two differently written regular expressions, whose meaning are the same, must return the same result.
- The system should give consistent results with user's search options.

3.2.2.3 Availability

- The IMERS shall be available 24 hours in a day in the device it is installed.
- The mean time between two occurrences of a same kind of service failure (MTTF) should be at least a week of usage.
- Whenever the system crashes, the mean time to restore service (MTRS) is expected to be hundreds of milliseconds.
- Informative messages should pop up whenever any components of the system have difficulty to respond to user actions.

3.2.2.4 Redundancy

- There is no actively redundant assets that are ready to replace their counterparts, since each component of the system are specialized in their functionality and no need to back up any of them to an already working component whose functionality is different since the system failure risk is low.
- There is no extra passively redundant assets for the services that may be interrupted. However, the CLT of the system could be used in case something go awry in the UI of the external interfaced system.

3.2.2.5 Error-Handling

- When a component of IMERS with UI is temporarily unavailable, the user shall be conducting his/her search from command line tool.
- The system allows the user to roll back any transactions before they have been committed whatsoever.
- The system prevents incorrect regex from being entered by the user. For instance, whenever two operands are written contiguously and searched, an error shall be given to the user to make him/her enter a correct regex.

3.2.3 Performance

This section of non-functional requirements describes requirements related to speed and efficiency. These are examined in five subsections; namely, stress, turnaround time, response time, throughput and, startup and shutdown.

3.2.3.1 Stress

- There is no need to concurrent access of multiple users to the system, since the program is built upon a single user system.

3.2.3.2 Turnaround Time

- This time is calculated by comparing the system to the similar systems.
- It is expected that this system will give the results of any query shorter than the systems resembling to this when there are a huge number of documents. This comparison would lose its discernibility when the document number drops. Since the biggest advantage of this system is to use an intelligent type of index which decreases the number of "to be searched documents". It may show no difference when compared to its peers when there is only a simple file to search for.

3.2.3.3 Response Time

- The maximum allowable time that a user must wait for any response is bounded by turnaround time from above.
- The minimum allowable time that a user must wait for any response is bounded by several milliseconds from below, which occurs in cases of any service failure notification or error-handling situations.

3.2.3.4 Throughput

- Throughput depends on the number of documents provided and the regex pattern as well. User may give thousands of files to search for and there would be hundreds of search results; moreover, user may write a popular regex pattern that can be encountered at every paragraph of every page, this would result a tremendous amount of documents, which means huge throughput.

3.2.3.5 Startup and Shutdown

- Program opening and closing delays are dependent on operating system.
- To open the IMERS with UI, user has to wait the user interface to be loaded onto the screen as well as that user has to wait the user interface to be unloaded from the screen.

3.2.4 Supportability

This section of the non-functional requirements describes requirements related to ability to monitor and maintain the system. The requirements that makes the system supportable are examined in seven subsections; namely, scalability, expected

changes, maintainability, configurability, localizability, installability and compatibility.

3.2.4.1 Scalability

- The system can be enlarged to be used over millions of documents.
- It can be used on web along with the integration to a database management system.

3.2.4.2 Expected Changes

- System is originally not implemented for online search purposes. However, this does not mean at the future there would not be an online web page searching imers. This change may mean a change in the implementation of the parsing part as well as buying a domain name for the website that imers be giving its service online.

3.2.4.3 Maintainability

- The design choices and updates should be documented on a regular basis.
- The documents should be well prepared and easy to reach to correct deficiencies quickly.
- All design aspects should be well integrated with each other in order to increase the understandability.
- The requirements written in this document, and the possible changes in later versions should be followed in design and construction phases of the software development process of this product.
- To maximize the useful life of the product, the graphical user interface should be kept as simple as possible.

3.2.4.4 Configurability

- Since the system is designed by using low-level technologies and does not require high-level software to run, it is easily adjustable.

3.2.4.5 Localizability

- Since the functionality of the product is pretty obvious and does not require a user to know different languages other than regular expression grammar provided in manual page and English, it does not need to support any local conditions or requirements.

3.2.4.6 Installability

- The system is easily installable with both downloaded debian packets and through Linux command line.

3.2.4.7 Compatibility

The system is compatible with both 32-bit and 64-bit versions of Linux based operating systems, such as; Ubuntu, Mint, Pardus, CentOS, etc.

4. Data Model and Description

This section includes a class diagram delineating data objects and their relationships.

4.1 Data Description

In IMERS project, there exists three types of data processed by the components of the system. The first one is the regular expression being entered by the user. By the time a regular expression provided by user is being searched, the major components of the system processes the data in a sequential manner. To touch it a little bit elaborately in this section; at first place, query parser parses the regular expression, then, plan generator logically resolves and simplifies the bunch of parsed data and physically evaluates the usefulness of the data, and last but not least, the execution engine matches the resultant data of plan generator with the index lookup and in-memory matching. To be looked up index structure is constructed by the second type of data which is about to be mentioned now. The second one is the environment consisting of textual documents from which the index structure is constituted. These type of data is again provided by the user by choosing a path over which this operation is performed. The third and the last type of data is statistical data kept according to user's previous searches. This is used for getting more accurate results from upcoming searches.

4.1.1 Data Objects

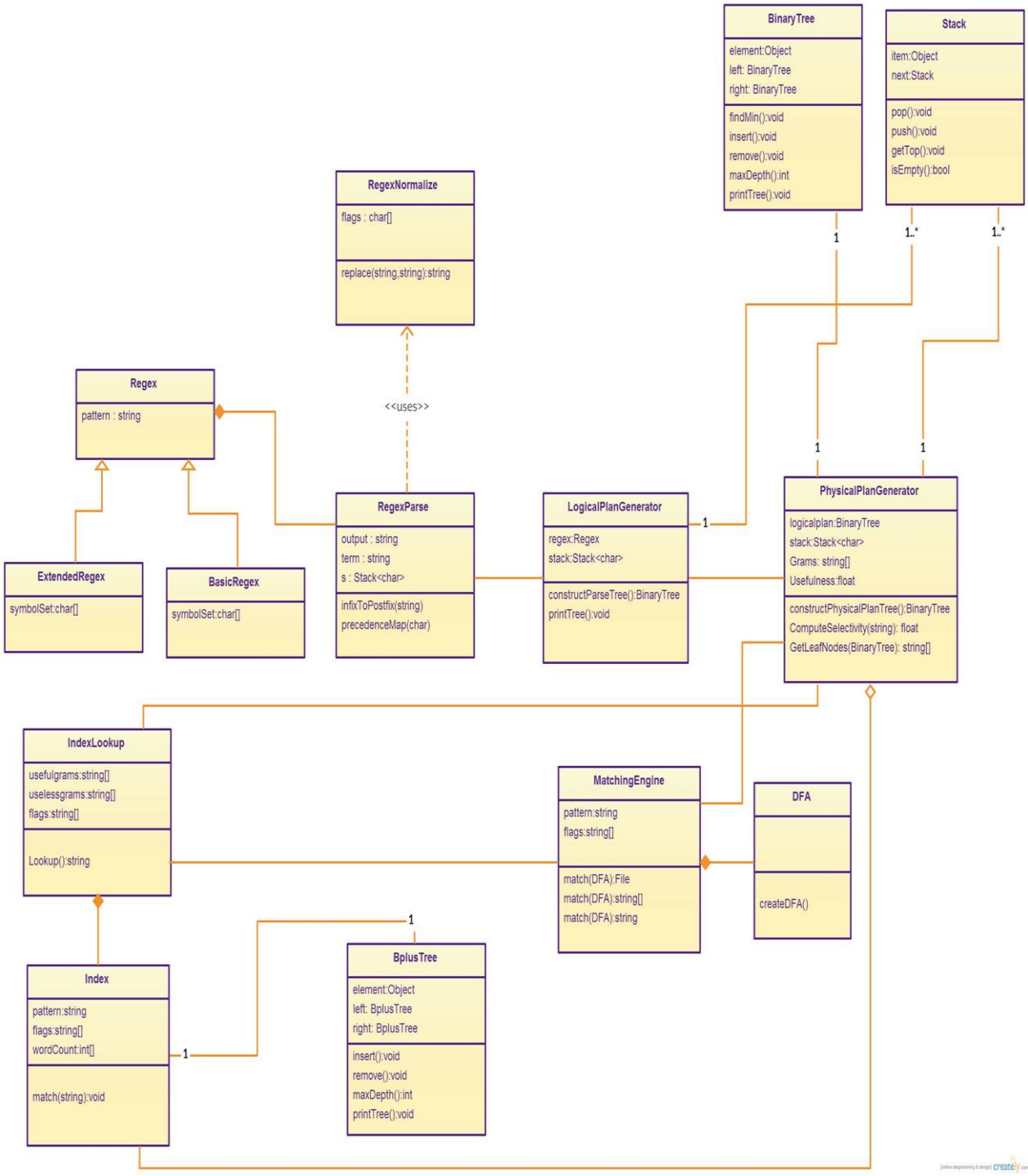

Figure 6: Class Diagram

The structure of the system including both data objects and the conceptual models of components as objects, their attributes, operations and relationships among each other are shown in the following class diagram.

- Regex: This class involves the data objects being used by RegexParser. pattern attribute denotes the string pattern entered by the user. It is also the superclass of BasicRegex and ExtendedRegex classes.

- BasicRegex: This class refers to regular expressions complying with the rules of being a basic regular expression. See 2.1.1.1.3 for example usage.

- ExtendedRegex: This class refers to regular expressions complying with the rules of being a extended regular expression. See 2.1.1.1.3 for example usage.

- RegexNormalizer: This class involves all the operation to convert entered regex into a form that only contains words, parentheses, and "|", "&", "*" as operands.

- RegexParser: This class uses the results of RegexNormalizer class as its term attribute, uses its infixToPostfix method, which needs s, an attribute of type stack, and benefits from the guidance of precedenceMap method, and writes the result string into its output attribute.

- LogicalPlanGenerator: This class involves the required operations to construct and show the parse tree from the output coming from RegexParser.

- PhysicalPlanGenerator: This class involves attributes related to gram keys coming from LogicalPlanGenerator. Its main purpose is to sustain selectivity operations and construct filtered physical plan tree.

- BinaryTree: This class is for a binary tree data structure to be used in several classes such as LogicalPlanGenerator.

- Stack: This class is for a stack data structure to be used in several classes such as RegexParser.

- IndexLookup class: Has three attributes and one method. It returns the path of the file which has the pattern the user provided. It has to hold the information about the useful and useless grams since it will look for the existence of useful grams first, in the case of failure, the only method "Lookup" will try to look for the combination of useless grams as well. If this fails, Lookup will return an empty string indicating no specific path found. This situation means that the useful grams were not enough to decrease the number of searched documents; therefore, matching engine will try to search from the start to find the patterns among all the files.

- Index class: Basically holds the index information and index structure that is constructed. IndexLookup class has this one to use it to find the grams. Index class is a class only for show. It is constructed by the help of flags, count of all the words to apply apriori algorithm, which allows PhysicalPlanGenerator class to make selectivity algorithm calculation. Lookup method in the IndexLookup class goes through the branches until to leaves to find the grams. BplusTree class is used while the index structure is created, that's why it has an association with that class.
- BplusTree: Is a data structure type appropriate to construct an index, it has the attributes that a tree data stricter has, also the methods needed are there as well.
- Matching engine: If the path is wanted, it returns the path directly. If the file name is wanted, it also returns the file name along with path directly. However, if the content of the file is wanted, it uses the deterministic finite automaton to match the string. It will try to apply the created DFA to find the pattern in all of the lines of the file. It goes sequentially among the files if there is any left. Otherwise it returns founded strings. If there is not any matches it gives no match found message to the user.

5. References

[1] IEEE Guide for Software Requirements Specifications," in IEEE Std 830-1984 , vol., no., pp.1-26, Feb. 10 1984, doi: 10.1109/IEEESTD.1984.119205,

[2] Appendix C of Don Widrig, Dean Leffingwell, "Managing Software Requirements: A Unified Approach," Addison-Wesley Professional, Release Date: October 1999, ISBN: 0201615932,

[3] http://www.gnu.org/software/grep/manual/grep.html

[4] http://wonconsulting.com/~cho/papers/cho-regex.pdf

[5] https://github.com/google/codesearch