**Part 2: Design:**

The search engine for Kingston University of Technology has been developed with the help of C# and lucene library. Lucene is a full-featured text search engine library written in Java and with the help of Lucene, it is really easy to develop a search engine. Lucene offers great features like scalability, high performance indexing and efficient search algorithms. Given below are the major classes used in our project. They are:

1. Program.cs

2. Stemmer.cs

3. Tokenisation.cs

4. UserSpecifiedSimilarity.cs

5. Form1.cs

6. Form2.cs

Program.cs is the starting point of the program, it has the Main() method and it is where the windows form gets displayed. Form1.cs and Form2.cs are the most important classes because it contains code for all the human computer interface and all processes starting from indexing happen in those two classes. Stemmer.cs contains the five steps used for stemming words and finding out the root words, which is done before indexing as well as before searching the information need given by user. Tokenisation.cs contains code for converting upper case letters to lower case letters and convert sentences into individual tokens.

**Major methods:**

**Main ():**

This is where the application gets started. Form1.cs is called from the Main () method, where the windows form will be displayed.

**Preprocessing ():**

This method is available in Tokenisation.cs class which is used to split the text into individual tokens. It splits the text using splitters like blank space, hyphen, new line character etc.

**StemTokens ():**

After converting the text into tokens, it is sent to stemTokens() method in Tokenisation.cs class. The tokens are then sent to Stemmer.cs class where it is stemmed with the help of five methods which are step1 (), step2 (), step3 (), step4 () and step5 ().

**CreateAnalyser ():**

Analyser is created in this method. Analysers process text linguistically and Lucene comes with multiple analysers.

**CreateWriter ():**

This method is used to create Index Writer which is used to write documents to the index.

**OpenIndex ():**

After getting the index location (directory location where the user wants to create index), we are using the Lucene.Net.Store.FSDirectory.Open() method to open the index in that directory.

**IndexText ():**

This method is used to create multiple fields for documents and index all the contents into those fields.

**CreateParser ():**

This method is used to create a Parser, which will parse all the text from documents.

**CreateSearcher ():**

This method is used to create the index searchers which will search for query in the index and retrieve the relevant documents. It is created using Lucene.Net.Search library.

**DisplayResults ():**

This method is used to display the retrieved documents in the decreasing order of relevance.

**Indexing strategy:**

Indexing is done with the help of snowball analyser which does all the functions of standard analyser with stemming as well. A stopword list is initialised with around 540 stopwords and used while creating the analyser. CreateAnalyser (), CreateParser (), CreateWriter (), OpenIndex () and IndexText () methods mentioned above are used to do indexing.

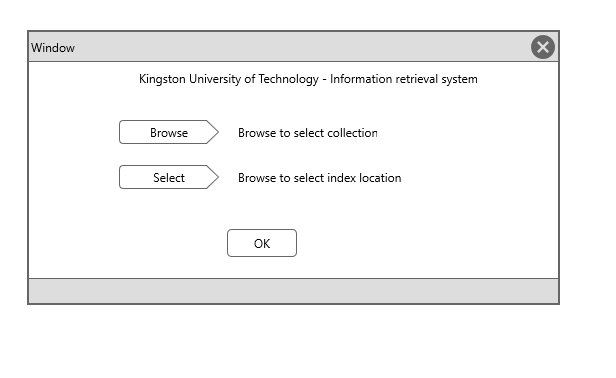
**Index structure:**

The contents of the documents are indexed using five fields, which are “DocID", "Title", "Author", "info", "first”. The contents are separated using the delimiters 'I', 'T', 'A', 'B', 'W'. After separating the contents using delimiters, the appropriate sections are indexed under respective fields.

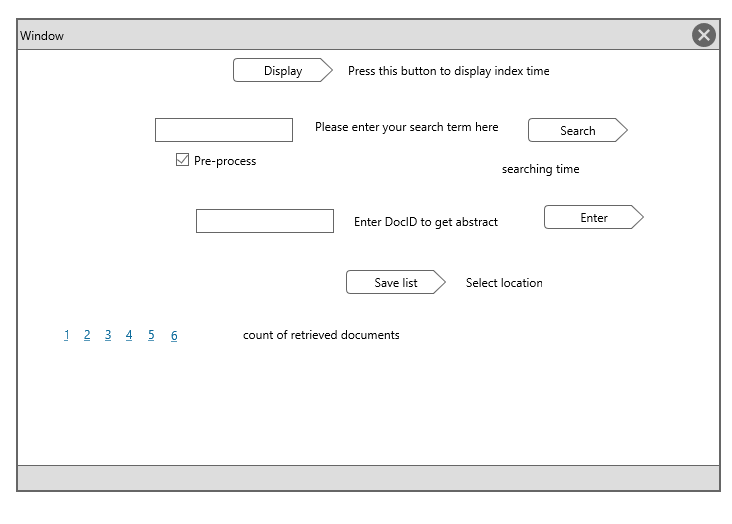
**Search strategy:**

Searching can be done with other options, one with pre-processing which includes converting all the letters to lower case, removing stop words and stemming. The other option is to search for the information need entered by the user without modifying anything in it. When pre-processing option is selected, the snowball analyser is used to do all the pre-processing and when pre-processing is not selected, standard analyser is used. Like mentioned above, createSearcher () and searchIndex () are two methods which are primarily used for searching.

**Mock-ups:**



This is the first windows form displayed when we run our application. Here, the user has to enter two directory paths. The first one is for selecting the location of the documents (corpus) and the second one is for choosing the location to store the index. On clicking the “Browse” and “select” button, the “Browse for folder” dialog box appears where the user can select directory paths they want. Once the directory paths are selected, the user can press the “OK” button.

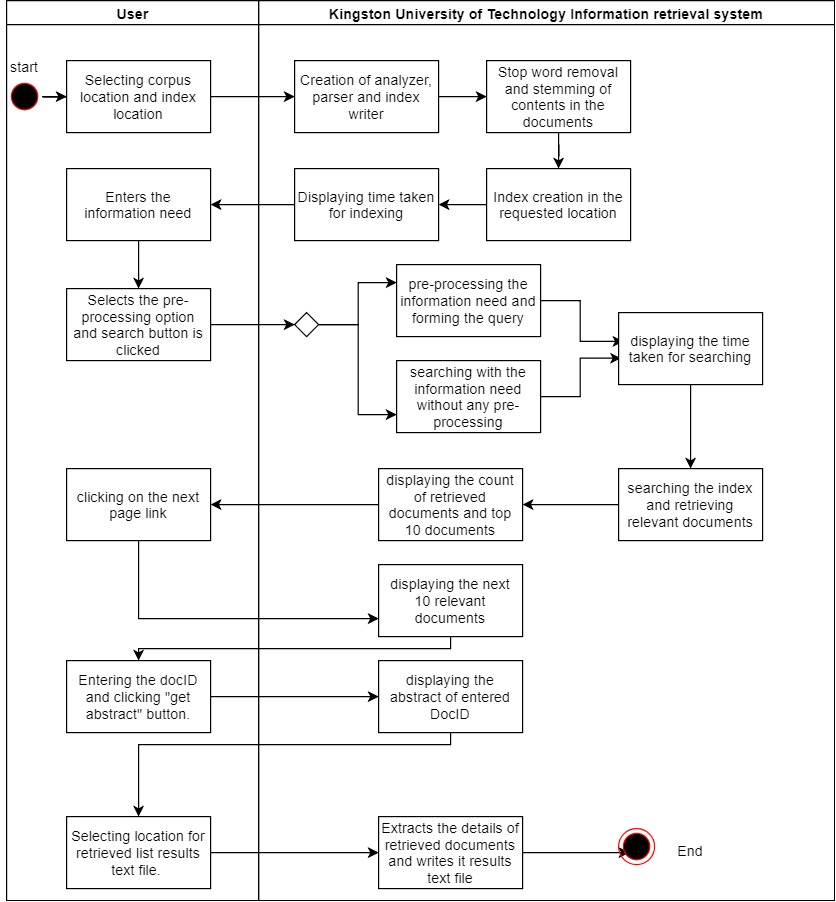


Once the “OK” button is pressed in the first dialog box, the second windows form will get displayed. It has a “Display” button which pressed will display the time taken for indexing in milliseconds in a message box. Then, the user can enter his information need in the text box, select the check-box if pre-processing is needed and when the search button is pressed, top 10 results will be displayed. In the “searching time” field, the time taken for searching will be displayed and in the “count of retrieved documents” field, the total count of the retrieved documents will get displayed.

If the user wants to view the abstract of a document, he/she can enter the DocID in the text box, and press the “Enter” button. It will trigger a new message box and the abstract of the respective document will be displayed there.

“Save List” is pressed when the user wants to save the list of retrieved results in a text file. On pressing the button, a new dialog box will be displayed prompting the user to select the location where the text file should be created. If the file already exists, the records will be appended to it or if the text file doesn’t exist, a new file will be created and the results will be inserted.

**Activity diagram:**



**Part-4: System Evaluation:**

**Efficiency metrics of new application:**

***Time to search for information needs:***

*001 – 149 milliseconds*

*002 – 17 milliseconds*

*023 – 14 milliseconds*

*157 – 19 milliseconds*

*219 – 37 milliseconds*

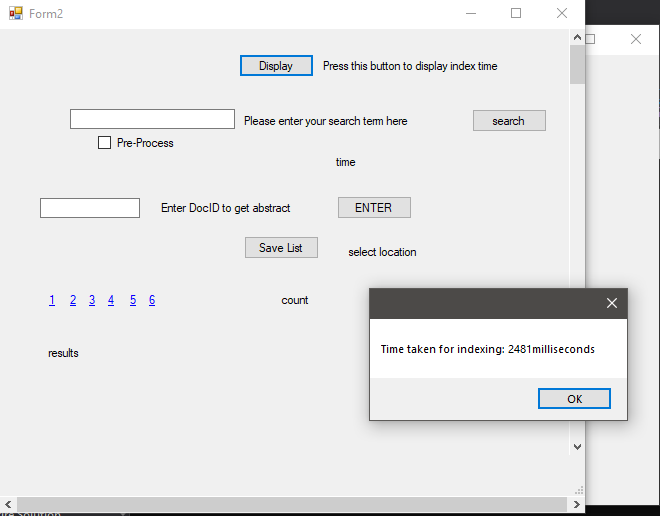
*Average search time for 5 queries: 47.2 milliseconds*

|  |  |
| --- | --- |
| **Metrics** | **values** |
| Index size | 838 KB |
| Time to produce the index | 2.168 seconds |
| Average time to search for queries | 0.0472 seconds |

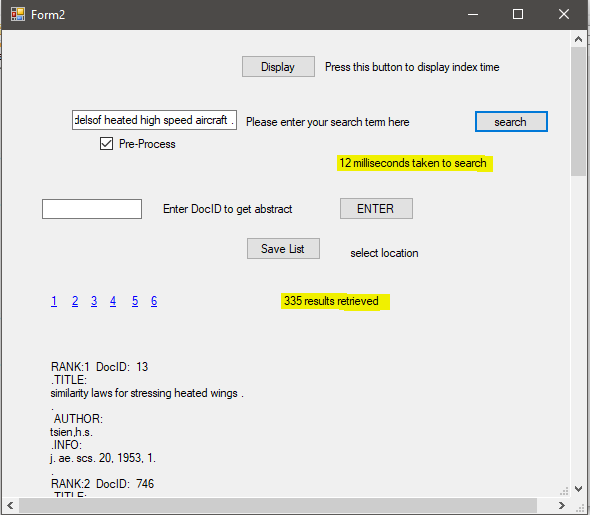
**Effectiveness metrics of new application:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***Topic*** | ***MAP*** | ***Precision*** | ***Recall*** | ***Precision @ 10*** |
| 001 | 0.2191 | 0.0597 | 0.6897 | 0.5000 |
| 002 | 0.1081 | 0.0357 | 0.4000 | 0.2000 |
| 023 | 0.0442 | 0.0415 | 0.7273 | 0.0000 |
| 157 | 0.0055 | 0.0324 | 0.1500 | 0.0000 |
| 219 | 0.0021 | 0.0090 | 0.0526 | 0.0000 |
| ***All*** | ***0.0758*** | ***0.0357*** | ***0.4039*** | ***0.1400*** |

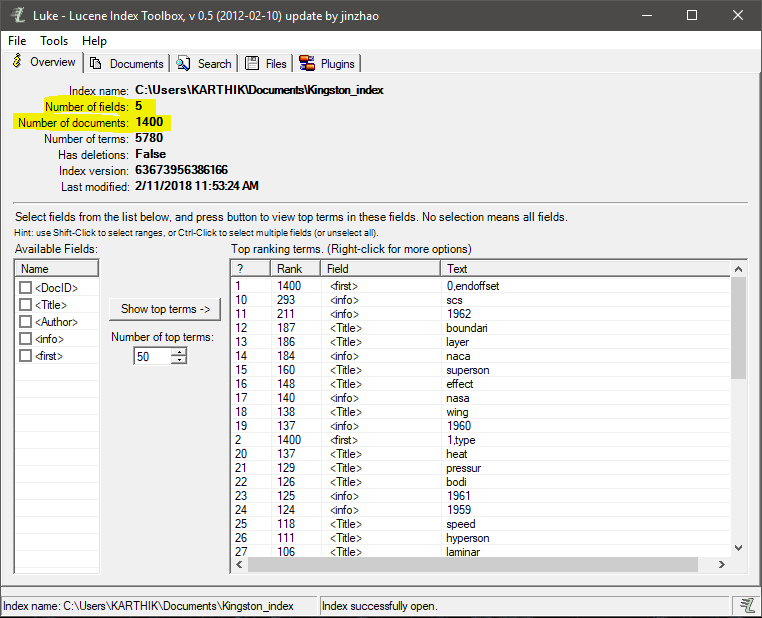
***Screenshots: Time taken for indexing***



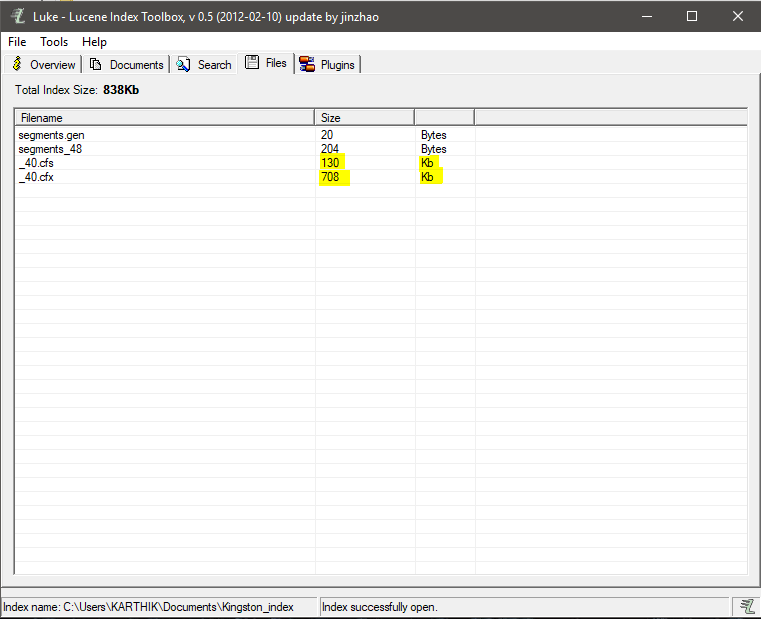
***Time taken for searching and retrieving documents:***

****

**Creating index:**

****

**Index size:**

****

**PART-5:**

**Comparison with baseline:**

|  |  |  |
| --- | --- | --- |
| ***Metrics*** | ***Performance of baseline system*** | ***Performance of new application*** |
| Index size | 4166 KB | 838 KB |
| Time to produce the index | 0.362 seconds | 2.168 seconds |
| Average time to search for queries | 0.0920 seconds | 0.0472 seconds |

***Interpolated recall-precision:***

|  |  |
| --- | --- |
| ***Recall*** | ***Interpolated precision*** |
| 0 | 0.4399 |
| 0.1 | 0.2102 |
| 0.2 | 0.1364 |
| 0.3 | 0.0930 |
| 0.4 | 0.0544 |
| 0.5 | 0.0383 |
| 0.6 | 0.0224 |
| 0.7 | 0.0087 |
| 0.8 | 0.0000 |
| 0.9 | 0.0000 |
| 1 | 0.0000 |

***Sign test:***

Here, we are comparing the performance of the new system to Baseline system.

|  |  |  |  |
| --- | --- | --- | --- |
| ***Query*** | ***Baseline*** | ***New System*** | ***Sign*** |
| 001 | 0.2247 | 0.2191 | -1 |
| 002 | 0.1028 | 0.1081 | 1 |
| 023 | 0.0422 | 0.0442 | 1 |
| 157 | 0.0028 | 0.0055 | 1 |
| 219 | 0.0020 | 0.0021 | 1 |

The new system performed better overall and it has outperformed baseline system on 4/5 queries.

Using the raw differences in metrics:

|  |  |  |  |
| --- | --- | --- | --- |
| ***Query*** | ***Baseline*** | ***New System*** | ***Raw difference*** |
| 001 | 0.2247 | 0.2191 | -0.0056 |
| 002 | 0.1028 | 0.1081 | 0.0053 |
| 023 | 0.0422 | 0.0442 | 0.0020 |
| 157 | 0.0028 | 0.0055 | 0.0027 |
| 219 | 0.0020 | 0.0021 | 0.0001 |
|  |  |  | ***Total = 0.0045*** |

The new system performed better overall, since the total of the signed ranks is positive.

***Wilcoxon signed ranks:***

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ***Query*** | ***Baseline (A)*** | ***New System (B)*** | ***B -A*** | ***Abs (B-A)*** | ***Rank*** | ***Signed rank*** |
| 001 | 0.2247 | 0.2191 | -0.0056 | 0.0056 | 5 | -5 |
| 002 | 0.1028 | 0.1081 | 0.0053 | 0.0053 | 4 | +4 |
| 023 | 0.0422 | 0.0442 | 0.0020 | 0.0020 | 2 | +2 |
| 157 | 0.0028 | 0.0055 | 0.0027 | 0.0027 | 3 | +3 |
| 219 | 0.0020 | 0.0021 | 0.0001 | 0.0001 | 1 | +1 |
|  |  |  |  |  |  | **W = 5** |

The new system performed better overall since the total of the signed ranks is positive.

**PART – 7:**

**ADVANCED FEATURES**

***1. Word associations:***

Words which are commonly used together should be found out from the queries. For example, if the user searches for “swimming” then the retrieved results should be related to “pools near the user”, “beaches near the user” and “lakes & ponds near the user”.

There are two kinds of associations:

1. ***Syntagmatic associations:*** It means the association between words which are related to each other.

Ex: University/courses

1. ***Paradigmatic associations:*** It means the association between words which can be potentially replaced by the other word.

Ex: zoos/wildlife safari

***Implementation:***

Word associations can be implemented with the help of equivalence classes. For example, if the user enters “U.S.” the retrieval system should retrieve documents which are related to “United States of America”, “U.S.A” as well as “America”.

In the same way, words which are related to each other or words which can be potentially replaced by other words are put in the same equivalence class.

“Univeristy/ courses/ program structure/ fee structure”. All these things can be put in the same equivalence class; hence if the user searches for “universities”, the information retrieval system will retrieve documents which are related to “courses”, “program structure” and “fee structure” as well.

This change should be done in the “searchIndex” method. Before, we are using the query to retrieve relevant documents; we can use the variance classes and get the associated words with the query. Then, we can add the associated words to the query and use it as the search term.

***2. Ontology based search:***

Keyword based retrieval system is a kind of information retrieval system where the relevant documents are retrieved based on keyword matching. But, this information system has a drawback which is, it won’t retrieve documents based on the meaning of the query but with the keyword only.

For example, if the user enters “houses for rent” in the search engine, he will expect all kind of documents such as “motels” , “hotels” and “Airbnb” to be retrieved. But, the keyword based information retrieval system will display documents which have the words “houses for rent” in it. In the case of ontology based information retrieval system, it will retrieve all the documents which are related to rental houses.

***Implementation:***

It can be implemented with the help of thesaurus. We should import “thesaurus” library into our project and send our query to it. Thesaurus searches for different words with the same meaning as your query and will return all those words to us. Then, we can add all those words to our query and do our searching process. It will result in retrieving the documents which not only have the same keyword but also the documents which have words with similar meaning.

This change should be implemented in the “searchIndex” method, where we can use thesaurus and find out all the words which have the same meaning as the words in query. Then, all those words which imply the same meaning can be used to retrieve documents.