B659 Search Assignment 1: Indexation
Suhas Jagadish

Task 1: Generating Lucene Index for Experiment Corpus (AP89)

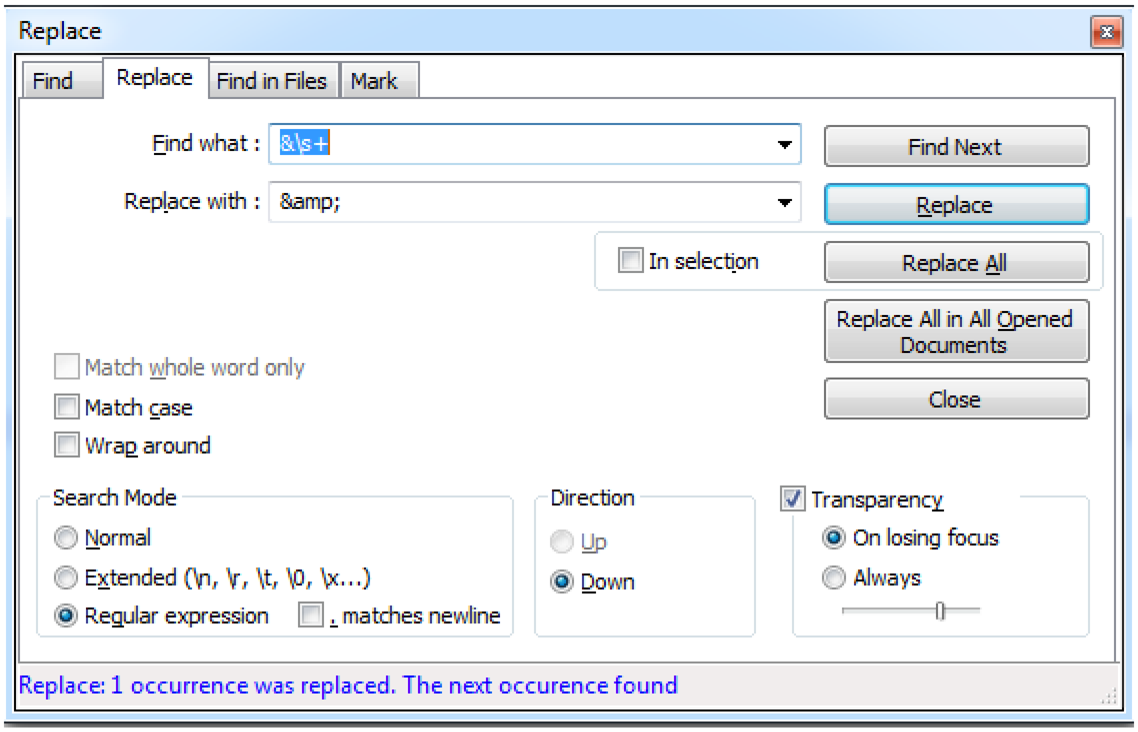
The code for this task is saved as “generateIndex.java” and “file_merge.java”.

File_merge.java: This file is used to combine all the .trectext files into one large document and appends the “root” tag required for the Java DOM parses. The reason I processed all the .trectext files into a single document is because the file processing and the program execution was fast when compared with processing individual files.

Also the in-built java xml parser expects a keyword when it hits “&” symbol. XML treats “&amp;” as “&” so whenever the parser finds just “&”, it will look for “&amp;” and throw an exception. In order to resolve this, I simply used regular expressions in notepad++. The regex is **&\s+**

Whenever we hit this regex, we replace the regex with “&amp;”

Please see the screenshot below on how I did this in notepad++ -



Find what : &\s+
Replace with : &amp;
[ ] In selection
[ ] Match whole word only
[ ] Match case
[ ] Wrap around
Search Mode: ( ) Normal ( ) Extended (\n, \r, \t, \0, \x...) [✓ Regular expression] [ ] . matches newline
Direction: ( ) Up [✓ Down]
[✓] Transparency: [✓ On losing focus] ( ) Always
Replace: 1 occurrence was replaced. The next occurence found

So please run the file_merge.java to generate an output file (It will be saved as out.txt), open it in notepad++ and make this minute change before running out actual “generateIndex.java” program. I sincerely apologize for the work-around, I couldn’t come up with a dynamic solution in time.

Also the filepaths in all the programs is absolute. For example, in this program,

```
File folder = new File("D:/Suhas/IUB/3rd Sem/Search/Assignments/corpus/corpus");
```

Please replace this path with your filepath before running the program.

*I have uploaded the actual output file(out.txt) in IU box, in case you don't want to do the above procedure. You can just use this file to run the generateIndex.java program. The link to download the file is https://iu.box.com/s/xnlxbesfhe9d3uf2xchknzhwcwndlgog

generateIndex.java: Once file_merge.java is called, please run this program to create indexes on five fields DOCNO, HEAD, BYLINE, DATELINE, and TEXT. Note that I have merged the contents from multiple elements if they bear the same tag.

Again absolute filepath is used in this program (Really sorry for not making the filepaths relative)

```
File fXmlFile = new File("D:/Suhas/IUB/3rd Sem/Search/Assignments/out_actual.txt");
```

```
Directory dir = FSDirectory.open(Paths.get("D:/Suhas/IUB/3rd Sem/Search/Assignments/Index"));
```

The above directory path should be specified to generate indexes.

(1) How many documents are there in this corpus? – **84474**

(2) Why different fields are treated with different kinds of java class? i.e., StringField and TextField are used for different fields in this example, why?

Solution: TextFields usually have a tokenizer so the indexed content is broken into separate tokens. Hence we define our "TEXT" tag as a TextField since we need to tokenize it and extract useful results. TextFields are generally useful for keyword search.

StringFields on the other hand do not have any tokenization. The entire String value is indexed as a single token. StringFields are generally used to index IDs(DOCNO in our case), phone numbers, zipcode etc, and are used to sort the documents, exact matches and to filter queries.

Task 2: Test different analyzers

The code for this task is saved as indexComparison.java. Again, I apologize for the absolute filepath in the program, which you have to change it to your system filepath.

Analyzer	**Tokenization applied?**	**How many tokens are there for this field?**	**Stemming applied?**	**Stop words removed?**	**How many terms are there in the dictionary?**
KeywordAnalyzer	No	84474	No	No	84061
SimpleAnalyzer	Yes	37316074	No	No	169981
StopAnalyzer	Yes	26202405	No	Yes	169948
StandardAnalyzer	Yes	26635610	No	Yes	233384

This program evaluates different analyzers and the results are displayed in the below table -

I ran the below code for a part of the text, for each analyzer, to determine if stemming is applied or not. I could see that the vocabulary terms had not stemming words applied for any of the analyzers.

```
//Print the vocabulary for <field>TEXT</field>
TermsEnum iterator = vocabulary.iterator();
BytesRef byteRef = null;
System.out.println("\n*******Vocabulary-Start**********");
while((byteRef = iterator.next()) != null) {
String term = byteRef.utf8ToString();
System.out.print(term+"\t");
}
System.out.println("\n*******Vocabulary-End**********");
```

For KeywordAnalyzer, we know that the number of tokens = numbers of keywords. This analyzer treats the entire content between <TEXT> </TEXT> as a single keyword/token. Hence we say that tokenization is not applied on Keyword Analyzer. But tokenization is applied on the rest of the analyzers.

Extra question: *can you index the documents via n-gram?*

Solution: Yes. The CommonGramsFilter class constructs bi-grams (in general n-grams) for frequently occurring terms while indexing. Basically this class looks for frequently occurring terms (2 terms together, 3 terms together …) and calculates the frequency distribution of these terms. Based on the frequently occurring terms, the indexer treats such n-gram terms as a single index and indexes the entire document.

I did not get the time to implement the functionality but this can be done as part of assignment 2.