

CSCI 5408

DATA MANAGEMENT AND
WAREHOUSING

ASSIGNMENT - 2

Banner ID: B00952865

GitLab Assignment Link:

https://git.cs.dal.ca/apurohit/CSCI5408_F23_B00952865_AdityaMaheshbhai_Purohit/-/tree/main/A2

Table of Contents

Problem 1A: Reuter News Data Reading & Transformation and storing in MongoDb.....	3
Pre-Requisite: Install MongoDb.....	3
Project Setup and Structure.....	7
Functional Testing.....	10
Flowchart.....	12
Algorithm.....	13
Problem 1B: Reuter News Data Processing using Spark.....	14
Pre-Requisite: Configure and initialize Apache Spark cluster in GCP.....	14
Explanation of task completion.....	16
Java Program.....	17
Functional Testing.....	20
Flowchart: Spark framework frequency count operation.....	23
Problem 2: Sentiment Analysis.....	24
Java Program.....	24
Functional Testing.....	29
References:.....	30

Problem 1A: Reuter News Data Reading & Transformation and storing in MongoDB.

Pre-Requisite: Install MongoDB

I have first installed MongoDB database [1] in my machine, so that my java program can store the new articles in MongoDB later on. Following are the steps that I followed:

Step 1: Go to the MongoDB website and download the setup for MongoDB Community edition.
<https://www.mongodb.com/try/download/community>

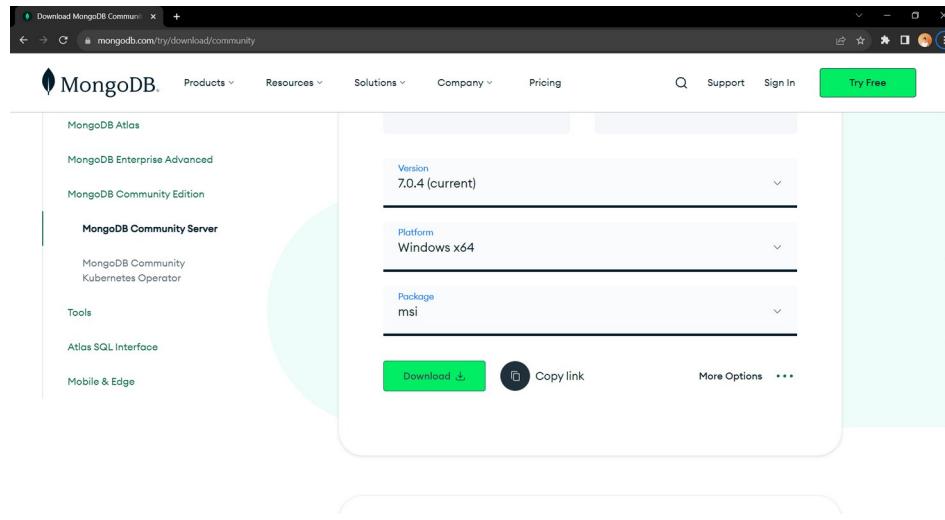


Figure 1: MongoDB Community Edition download page [1].

Step 2: Execute the MongoDB setup and click on next.



Figure 2: MongoDB Setup wizard welcome page [1].

Step 3: Accept the terms and click on next.

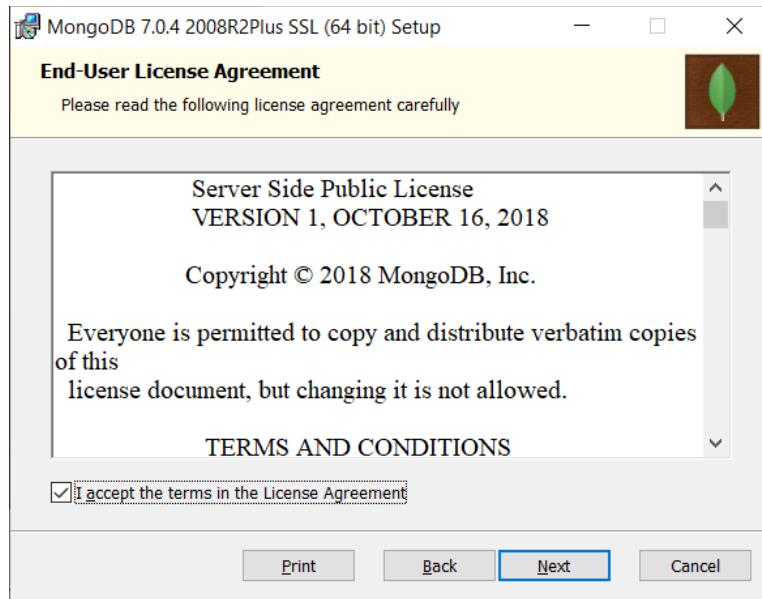


Figure 3: Accept the terms of MongoDB [1].

Step 4: Choose complete installation option, keep the default settings and click on next as show in figure 4 and figure 5.

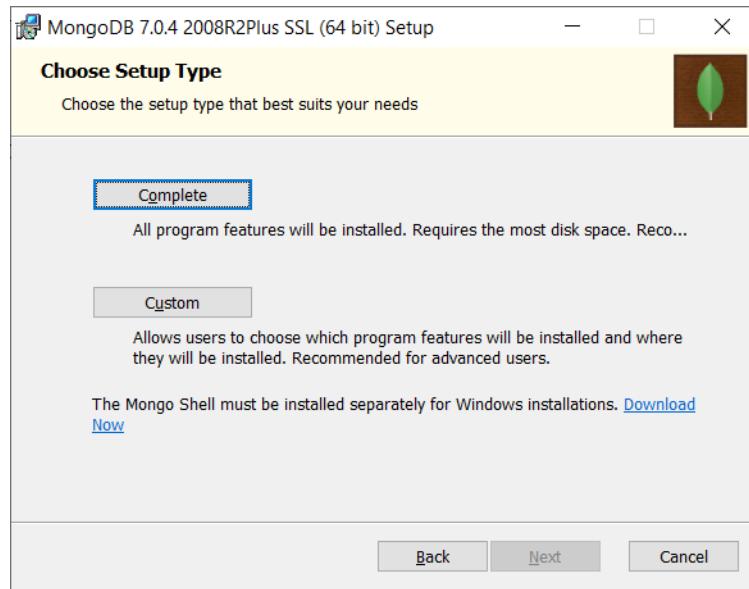


Figure 4: Choose complete installation [1].

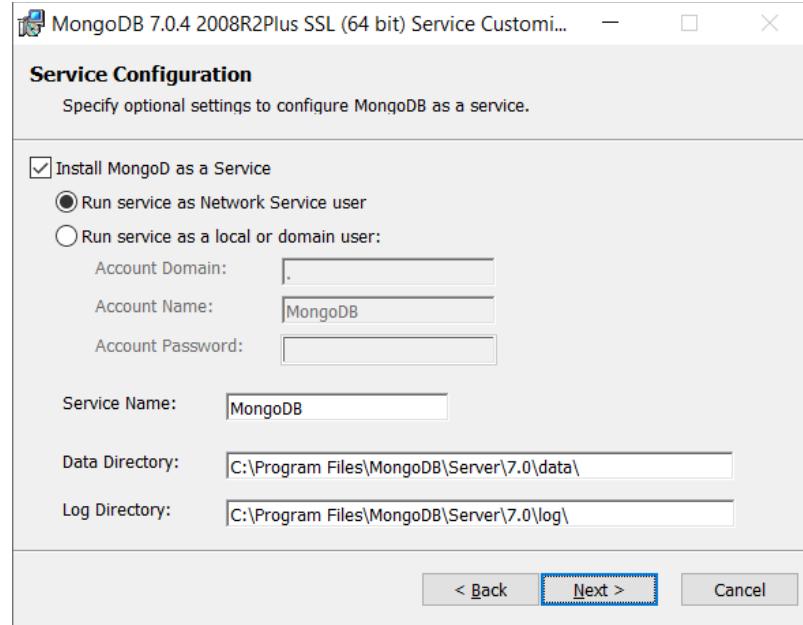


Figure 5: Default service configuration of MongoDB [1].

Step 5: Select MongoDB Compass installation and click on next. This is a graphical interface for MongoDB, which will be useful later on.

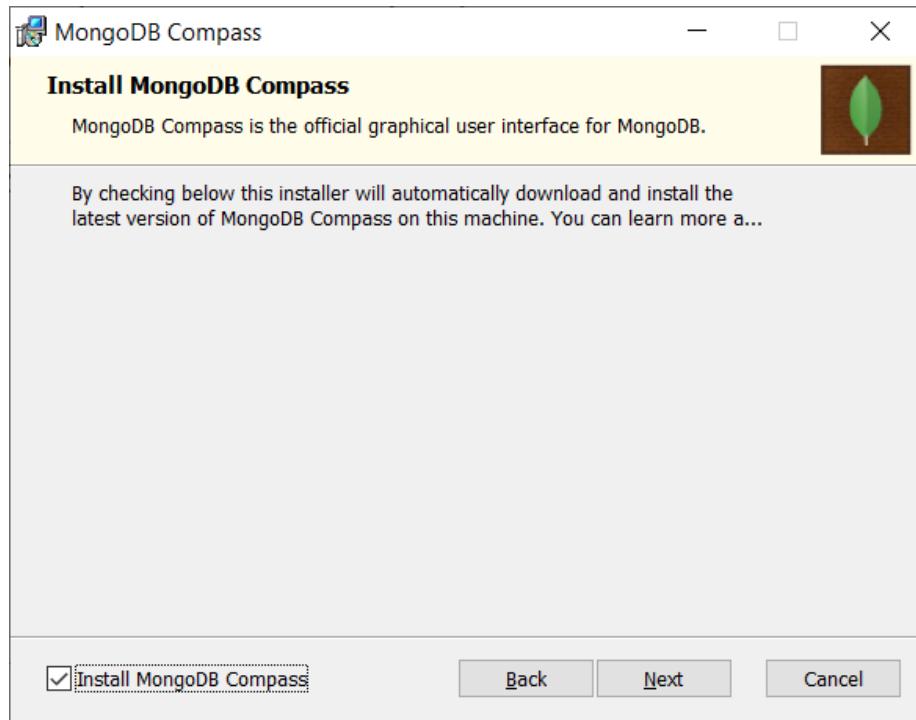


Figure 6: MongoDB Compass option selection [1].

Step 6: Click on install button and wait for the installation to complete.

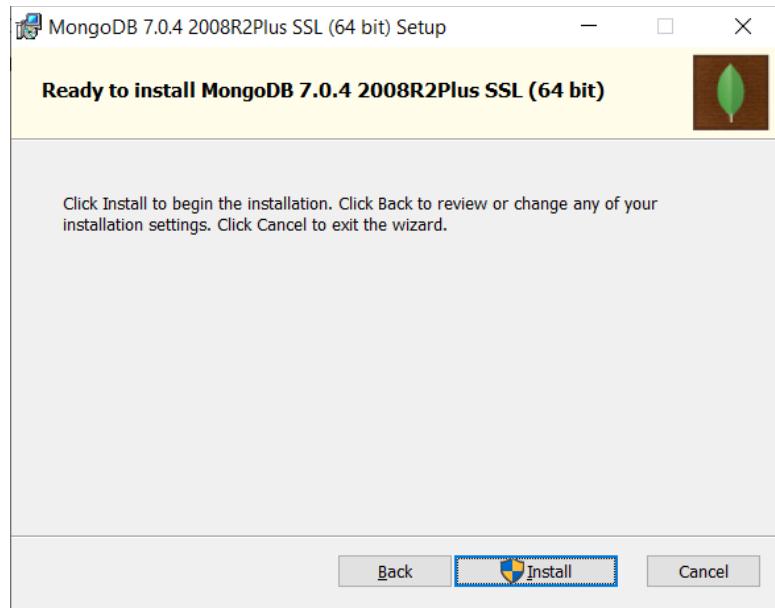


Figure 7: Final installation step for MongoDB [1].

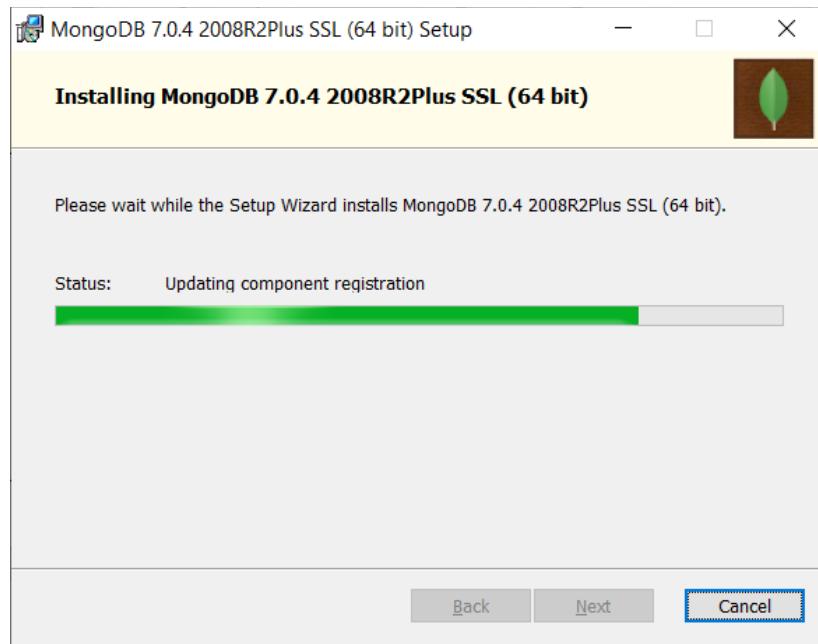


Figure 8: MongoDB installation in progress [1].

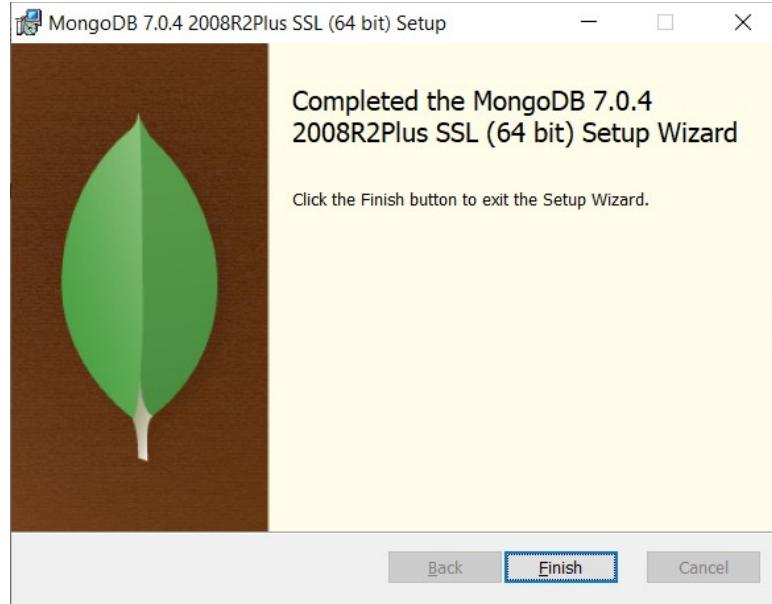


Figure 9: Installation completed for MongoDB [1].

Project Setup and Structure

I have created a java maven project which contains a main method class called “ReutRead” as shown in figure 10. Whole java code is commented as per JavaDoc commenting standards.

```

    import java.util.List;
    import java.util.regex.Matcher;
    import java.util.regex.Pattern;

    /**
     * The ReutRead class provides method to read news articles from *.sgm files,
     * extract title and body using regular expressions, and store that in MongoDB database.
     * The main method creates connection to MongoDB Server, creates a database and collection,
     * then adds all news articles from 2 .sgm files, to the database.
     */
    public class ReutRead {

        /**
         * Adds all news articles from a REUTERS file to a MongoDB database.
         *
         * @param fileName Path to the file containing news articles.
         * @param mongoDatabase A MongoDB database object, which is connected.
         */
        public static void addArticlesToMongoDB(String fileName, MongoDatabase mongoDatabase) { ... }

        /**
         * Entry point of the ReutRead Class. This creates a database connection and adds news articles to it using helper method.
         *
         * @param args Command line arguments.
         */
        public static void main(String[] args) { ... }
    }

```

Figure 10: ReutRead.java file and ReutRead Class [2] [3] [4] [5].

In this folder, I have kept both the .sgm files in the root directory as shown in figure 10. Moreover, in the ReutRead java class I have 2 methods: main() and addArticlesToMongoDB(). Here, the main method creates a MongoDB server connection, database and then invokes the addArticlesToMongoDB() method twice with 2 different .sgm files.

```

ReutRead src main java org example ReutRead
ReutRead - ReutRead.java
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ReutRead - ReutRead.java
ReutRead Project src main java org example ReutRead ReutRead.java
> idea
> src
> main
> java
> ReutRead
> ReutRead
> resources
> test
> target
> gitignore
> config
> reut2-009.sgm
> reut2-014.sgm
> External Libraries
> Scratches and Consoles
ReutRead.java
61 } catch (Exception e) {
62     System.out.println(e);
63 }
64 }
65 }

66 /**
67 * Entry point of the ReutRead Class. This creates a database connection and adds news articles to it using helper method.
68 *
69 * @param args Command Line arguments.
70 */
71 public static void main(String[] args) {
72
73     // Connect with MongoDB, create database and collection.
74     MongoClient mongoClient = new MongoClient("localhost", 27017);
75     MongoDatabase mongoDatabase = mongoClient.getDatabase("reuterDb");
76     mongoDatabase.createCollection("articles");
77
78     // Add articles from reut2-009.sgm file to ReuterDb mongoDB Database.
79     ReutRead.addArticlesToMongoDB("reut2-009.sgm", mongoDatabase);
80
81     // Add articles from reut2-014.sgm file to ReuterDb mongoDB Database.
82     ReutRead.addArticlesToMongoDB("reut2-014.sgm", mongoDatabase);
83
84 }
85
86

```

Figure 11: main() method logic of ReutRead Class [2] [3] [4] [5].

The addArticlesToMongoDB(..) contains the core logic of parsing title and body of news articles, as well as adding them to the ReutersDB named MongoDB database as shown in figure 12.

```

ReutRead src main java org example ReutRead ReutRead.java
ReutRead - ReutRead.java
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help ReutRead - ReutRead.java
ReutRead Project ReutRead ReutRead.java
> idea
> src
> main
> java
> ReutRead
> ReutRead
> ReutRead.java
28 public static void addArticlesToMongoDB(String fileName, MongoDatabase mongoDatabase) {
29     try {
30         // Read the file contents into a String.
31         BufferedReader bufferedReader = new BufferedReader(new FileReader(fileName));
32         StringBuilder sgmFileContentTemp = new StringBuilder();
33
34         String lineOfFile;
35         while ((lineOfFile = bufferedReader.readLine()) != null) {
36             sgmFileContentTemp.append(lineOfFile);
37         }
38         bufferedReader.close();
39
40         String sgmFileContent = sgmFileContentTemp.toString();
41
42         // Use regex to extract all text between all <TITLE> and <BODY> tags, under the parent tag: <REUTERS>.
43         Pattern titleAndBodyPattern = Pattern.compile("<REUTERS>.*<TITLE>.*<BODY>.*</TITLE>.*</BODY>.*</REUTERS>", Pattern.DOTALL);
44         Matcher matcher = titleAndBodyPattern.matcher(sgmFileContent);
45
46         // Create a list of article documents, where each document has an article title and body.
47         List<Document> documentList = new ArrayList<Document>();
48         while (matcher.find()) {
49             Document document = new Document();
50             document.append("title", matcher.group(1));
51             document.append("body", matcher.group(2));
52             documentList.add(document);
53         }
54
55         // Add all articles(list of documents) to the MongoDB.
56         mongoDatabase.getCollection("articles").insertMany(documentList);
57
58         // Print the success message with the count of articles added to the database.
59         System.out.println("***** " + documentList.size() + " articles added to the database *****");

```

Figure 12: addArticlesToMongoDB method's code overview [2] [3] [4] [5].

Explanation of addArticlesToMongoDB method: The lines 30 to 40, are reading the contents of file from the particular .sgm file using BufferedReader and then storing the whole content into a string variable. After that, a single Regular Expression pattern is used to extract the content between <TITLE> and </TITLE> as well as between <BODY> and </BODY> tags. These tags should be under the <REUTERS>...</REUTERS> tags to get parsed properly. In the regex, the (?i) is used for case insensitive comparisons. Then we look for <REUTER (some attributes for this tag)> kind of a pattern using <REUTERS[^>]*>, then we look for a starting tag <TITLE> with some optional space characters(i.e. space, tab or newline character.) or non-space characters (i.e. a-z, A-Z, etc) which maybe present before the start of this tag. This is achieved using the /s|S)*?<TITLE> portion of the regex. Then we look for the end tag </TITLE> and extract whatever is before </TITLE> tag and after the <TITLE> tag. This is achieved using (/s|S)*?)</TITLE> part of regular expression. The round parenthesis, will help us to extract the text between these <TITLE> </TITLE> tags.

Then the same extraction is performed with the body tags as well using the /s|S)*?<BODY>(/s|S)*?)</BODY> part of the regex. Finally, we check for the ending tag </REUTERS>, to conclude the extraction process for a single news article. This matching process will happen for whole string, until we extract all the news articles.

For each article parsed, each title and body is added as a MongoDB document and then this document is then added into a list of documents. Finally, the whole list of documents is added into the articles collection of MongoDB database ReuterDb. A message is also displayed in the end to show how count of articles parsed from the current file and added to database.

The program explanation is also explained in terms of flowchart and algorithm later on in this report.

Functional Testing

I have kept the reut2-009.sgm and reut2-014.sgm files in the root directory of my java maven project before the execution of program. Then, I execute the main method of my java program.

```

    /*
     * Entry point of the ReutRead Class. This creates a database connection and adds news articles to it using helper methods.
     */
    public static void main(String[] args) {
        // Connect with MongoDB, create database and collection.
        MongoClient mongoClient = new MongoClient("localhost", port:27017);
        MongoDatabase mongoDatabase = mongoClient.getDatabase("ReuterDb");
        mongoDatabase.createCollection('articles');

        // Add articles from reut2-089.sgm file to ReuterDb mongoDB Database
        ReutRead.addArticlesToMongoDB({ fileName: './reut2-089.sgm', mongoDatabase });

        // Add articles from reut2-014.sgm file to ReuterDb mongoDB Database.
        ReutRead.addArticlesToMongoDB({ fileName: './reut2-014.sgm', mongoDatabase });
    }

```

Run: ReutRead
 INFO: Monitored thread successfully connected to MongoDB ServerDescription{address=localhost:27017, type=STANDALONE, state=CONNECTED, ok=true, version=ServerVersion{major=2, minor=0, build=137}, maxWireVersion=6, minWireVersion=6, maxDocumentSize=16777216}
 INFO: Found connection {connectionId=10, serverName=127.0.0.1:27017}
 INFO: Inserted document {connectionId=10, serverName=127.0.0.1:27017}
 **** Successfully added 984 news articles into MongoDB Database 'ReuterDb' from file ./reut2-089.sgm
 **** Successfully added 656 news articles into MongoDB Database 'ReuterDb' from file ./reut2-014.sgm
 Process finished with exit code 0

Figure 13: Execution of ReutRead program [2] [3] [4] [5].

Expected Result: Some articles should appear in the MonogDB database “ReuterDb”.

Actual Result: 1560 articles are added to the MongoFB database “ReuterDb”.

MongoDB Compass - localhost:27017/ReuterDb.articles

localhost:27017

Documents ReuterDb.articles

My Queries Databases +

Search

ReuterDb articles

Documents Aggregations Schema Indexes Validation

Filter Type a query: { field: 'value' } or [Generate query](#)

Explain Reset Find Options

1-20 of 1560

_id: ObjectId('6569e19848cfff7b359768')
 title: "ADVANCED MAGNETICS Ltd;ONGO IN AGREEMENT"
 body: "Advanced Magnetics Inc said it reached a four mln dlrs research and dev..."

_id: ObjectId('6569e19848cfff7b359768')
 title: "HEALTH RESEARCH FILES FOR BANKRUPTCY"
 body: "Rlt;Health Research and Management Group> said it has filed for protect..."

_id: ObjectId('6569e19848cfff7b359768')
 title: "THEREREX CORP Rlt;MRX> 2RD QTR JAN 31 LOSS"
 body: "The loss seven cts vs profit five cts Net loss 149,421 vs profit 18..."

_id: ObjectId('6569e19848cfff7b359768')
 title: "PU.S. SELLING 12.8 BILLION DLRS OF 3 AND 6-MO BILLS MARCH 30 TO PAY DOWN"
 body: "Commodore International Ltd said it settled and discontinued all pendin..."

_id: ObjectId('6569e19848cfff7b359768')
 title: "BALDIDGE SUPPORTS NIC TALKS ON CURRENCIES"
 body: "Commerce Secretary Nelson Baldridge said he supported efforts to persua..."

Figure 14: MongoDB Compass showing the 1560 articles added by program [1].

Test Conclusion: Passed

Flowchart

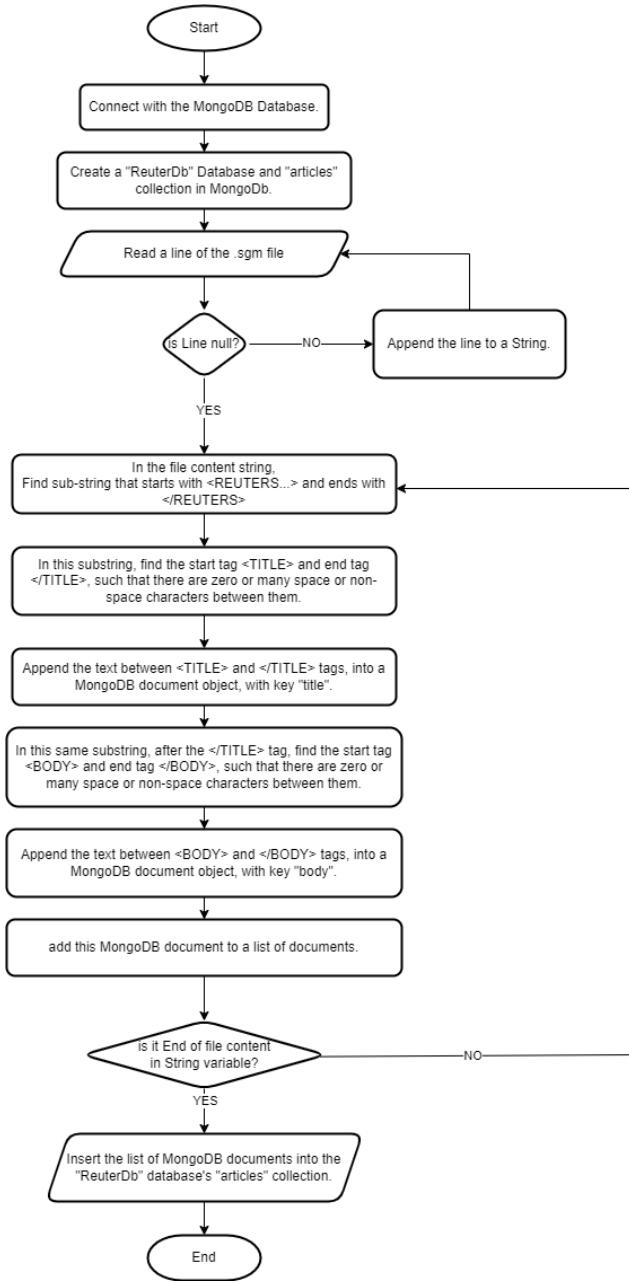


Figure 15: Flowchart of ReutRead program - Problem 1A [6].

This flowchart logic can be repeated for each .sgm file, whose content needs to be added to MongoDB. Moreover, in program, there is just a single regular expression that extracts both tile and body contents from the reuter new articles. However, for better understanding that regex logic is explained in multiple steps in the flowchart. From code perspective, regex pattern was explained in depth, in the “Explanation of addArticlesToMongoDB method” part of Project Setup and Structure section.

Algorithm

Step-1: Start.

Step-2: Connect with the MongoDB Community Server.

Step-3: Create “ReuterDb” database and “articles” collection in MongoDB.

Step-4: Read a single line from .sgm file.

Step-5: Check if the line is null or not, if not, append it to a String and go back to step-4.

Step-6: In the file content string, find sub-string that starts with <REUTERS...> and ends with </REUTERS>

Step-7: In this substring, find the start tag <TITLE> and end tag </TITLE>, such that there are zero or many space or non-space characters between them.

Step-8: Append the text between <TITLE> and </TITLE> tags, into a MongoDB document object, with key "title".

Step-9: In this same substring, after the </TITLE> tag, find the start tag <BODY> and end tag </BODY>, such that there are zero or many space or non-space characters between them.

Step-10: Append the text between <BODY> and </BODY> tags, into a MongoDB document object, with key "body".

Step-11: add this MongoDB document to a list of documents.

Step-12: If the file content is fully read, add list of documents to the MonogDB, else go back to step-6.

Step-13: If having more files, start again from Step-4, else go to Step-14.

Step-14: End

Problem 1B: Reuter News Data Processing using Spark.

Pre-Requisite: Configure and initialize Apache Spark cluster in GCP

Step-1: Go to Cloud Dataproc service in Google cloud platform using the search option and click create cluster.

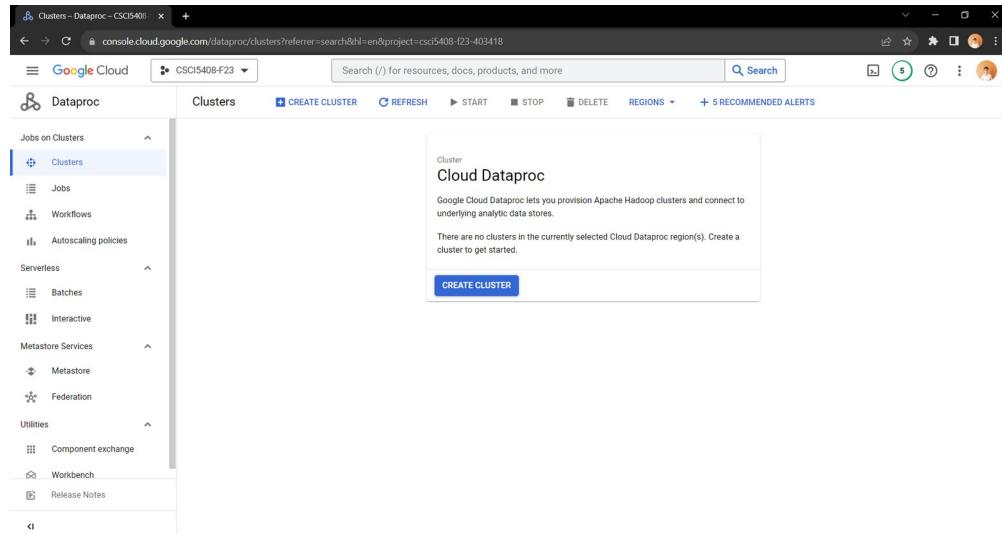


Figure 16: GCP Dataproc home screen [7].

Step-2: Give any convenient name to the cluster.

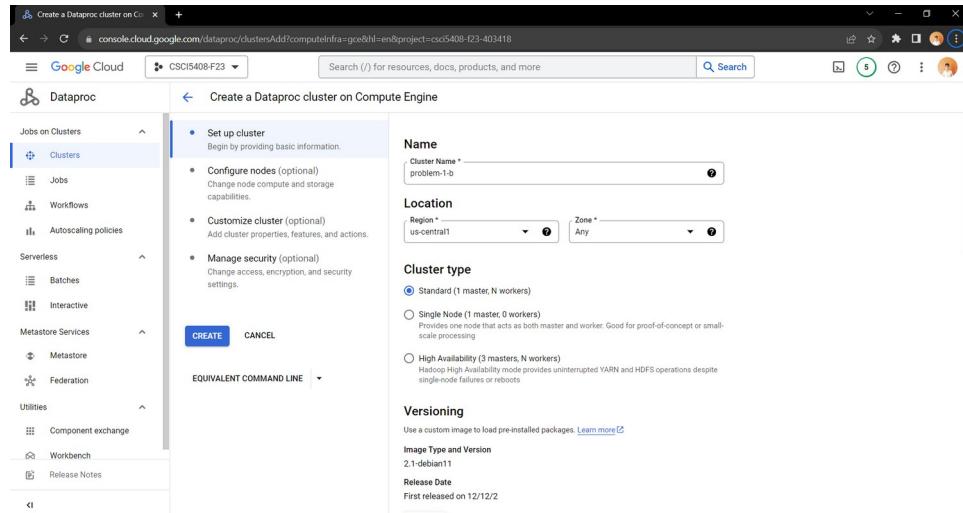


Figure 17: Creating cluster [7]

Step-3: Reduced the vCPUs of worked nodes to 2, due to usage limits of my account. Click create.

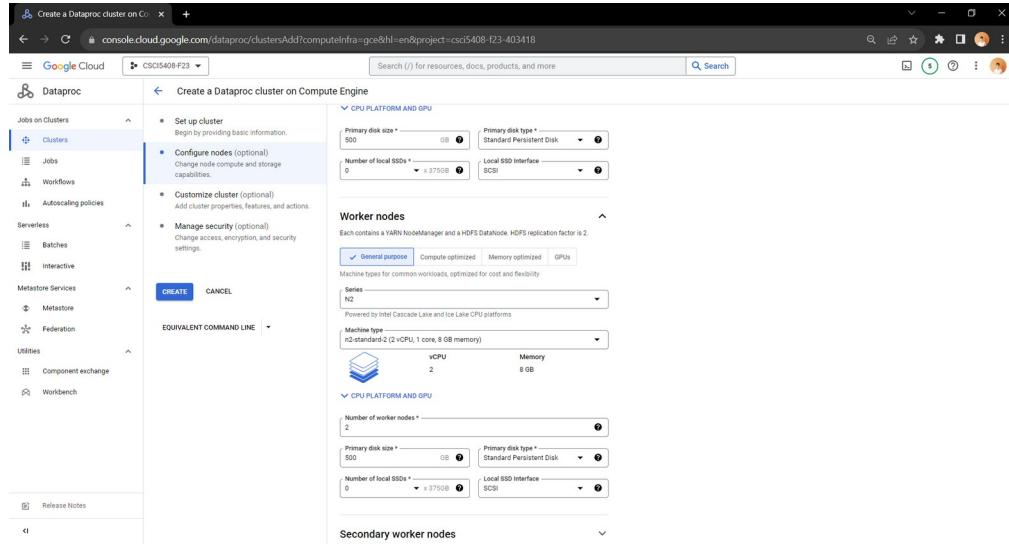


Figure 18: Worker nodes vCPU reduced [7]

Step-4: After a while, you should see all the nodes of cluster running.

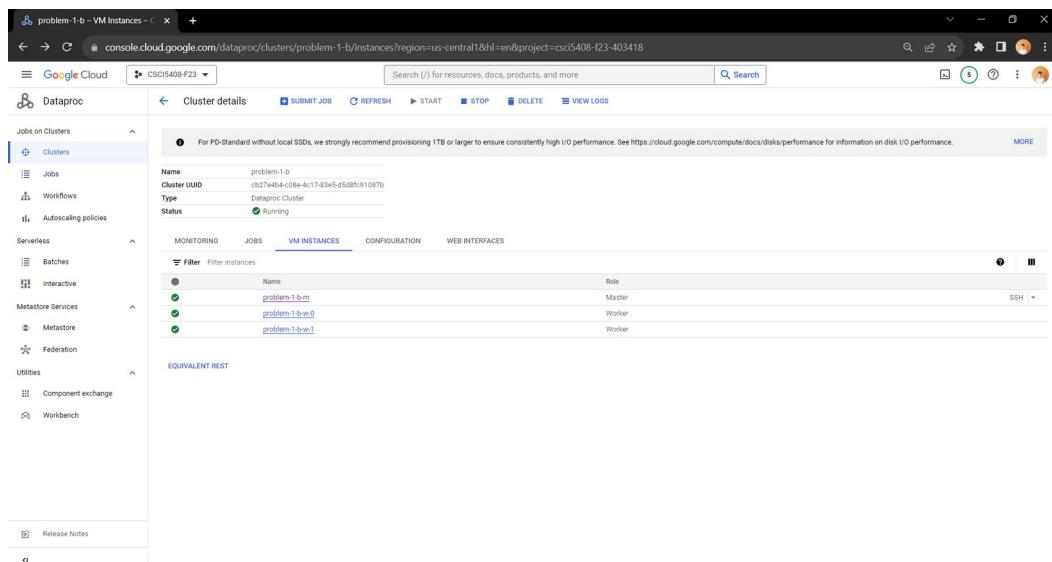


Figure 19: Cluster running [7]

Step-5: Connect to master node using ssh.

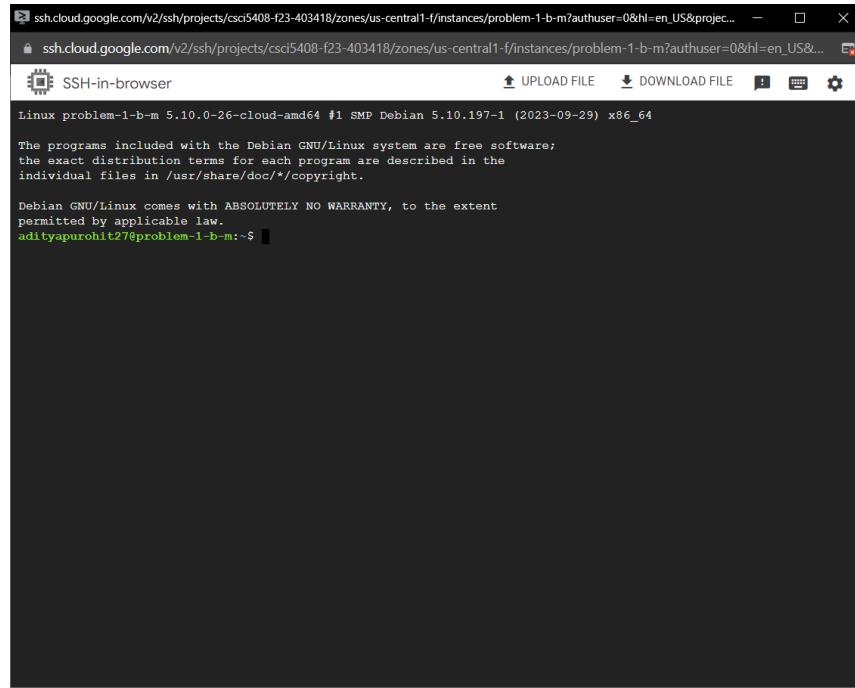


Figure 20: Connected to master node using SSH [7]

Explanation of task completion.

I logged into my GCP account and searched for the Dataproc service, so that I can create my Apache Spark cluster. I gave a name to my cluster as “problem-1-b” and reduced the count of vCPUs to 2 for the worker nodes, due to some account limitations. After that, I clicked on create cluster and waited for sometime till all nodes in the cluster started running. To test the connectivity with the master node, I used the ssh option, which worked fine.

Then, I created a java maven project, with a single .java file called ReutSpark.java. I also kept two more files in the root folder of the project, one called “stopwords.txt” [8] which has all stop words, which will be used for cleaning and another file “reut2-009.sgm” which has all the main data, in which the word frequency counting needs to happen.

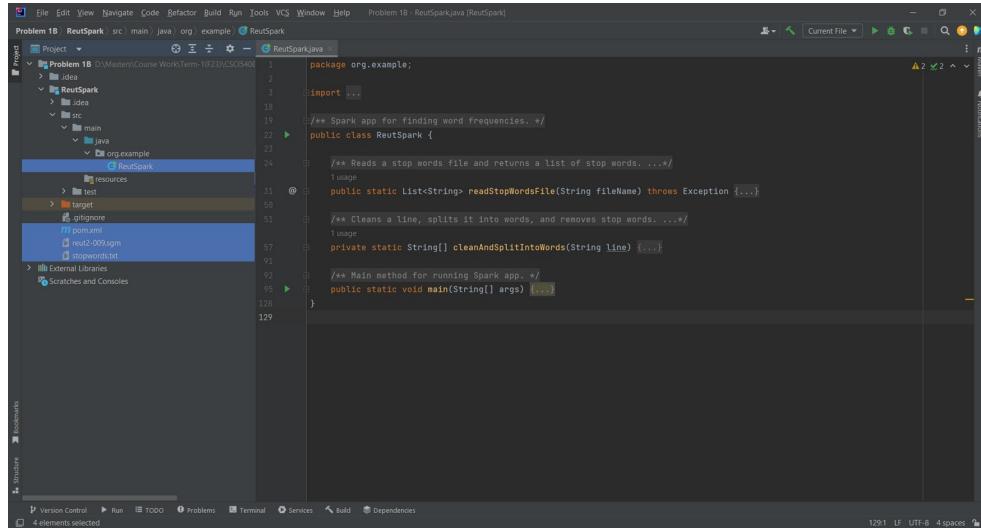
Overall, from code perspective, I first create a spark session and then load the whole .sgm file into a Spark dataset of lines. Then, for pre-processing, these lines are made upper case, cleaned (remove special characters and start & end tags), split into words and then finally the stop words are removed. After that, using groupBy function of Spark, the word frequency is calculated and displayed for each word. Finally, the highest frequency word and lowest frequency word is displayed with the help of descending and ascending sorting respectively.

Java Program

The java project structure looks something like shown in figure 21. There are 4 important files.

ReutSpark.java – This has all the core logic for word frequency. All parts of code are commented using Javadoc standards.

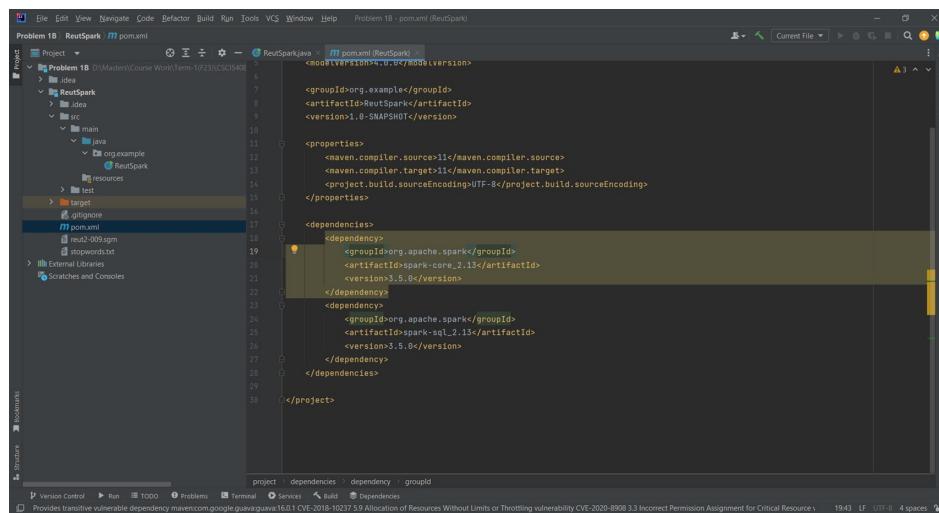
reut2-009.sqm is the input file for word frequency calculation.



The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure under 'Problem 1B'. It includes a 'ReutSpark' module with 'src' and 'test' sub-directories. Inside 'src', there's a 'main' directory containing 'java' and 'resources' sub-directories, which contains the 'ReutSpark.java' file. The 'ReutSpark.java' file is open in the main editor window, showing Java code for a Spark application. The code includes imports, class definitions, and methods for reading stop words and cleaning text. Below the editor, the status bar shows '1291 LF UFT-8 4 spaces'.

Figure 21: Overview of ReutSpark.java file and folder structure of project [2] [3] [4].

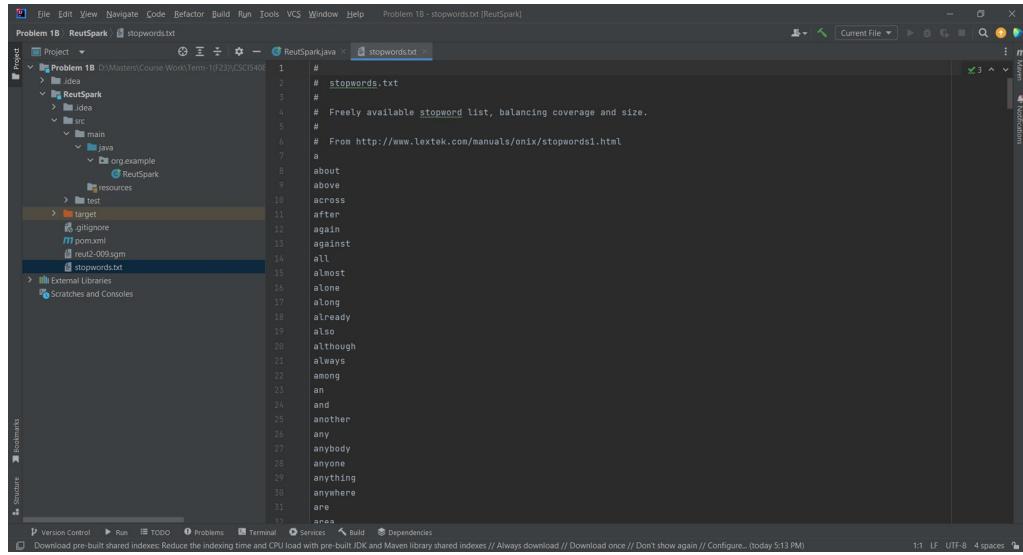
Pom.xml - 2 spark related dependencies are used in my maven project's pom.xml in order to use Spark related functionalities in java.



The screenshot shows the IntelliJ IDEA interface with the 'pom.xml' file open in the editor. The left sidebar shows the project structure with 'ReutSpark' selected. The 'pom.xml' file contains Maven XML configuration. It defines a group ID of 'org.example', an artifact ID of 'ReutSpark', and a version of '1.0-SNAPSHOT'. It includes properties for compiler source and target versions, and a build source encoding. The 'dependencies' section lists two dependencies: 'spark-core_2.13' and 'spark-sql_2.13', both from the 'org.apache.spark' group ID. The status bar at the bottom indicates a transitive dependency warning.

Figure 22: pom.xml file [2] [3] [4] [9] [10]

Stopwords.txt – This is a list of common stop words which are not much important and will be removed for our word frequency calculation. This file was downloaded from GitHub [8].



```

1  #
2  # stopwords.txt
3  #
4  # Freely available stopword list, balancing coverage and size.
5  #
6  # From http://www.lextek.com/manuals/onix/stopwords1.html
7  a
8  about
9  above
10 across
11 after
12 again
13 against
14 all
15 almost
16 alone
17 along
18 already
19 also
20 although
21 always
22 among
23 an
24 and
25 another
26 any
27 anybody
28 anyone
29 anything
30 anywhere
31 are
32 area

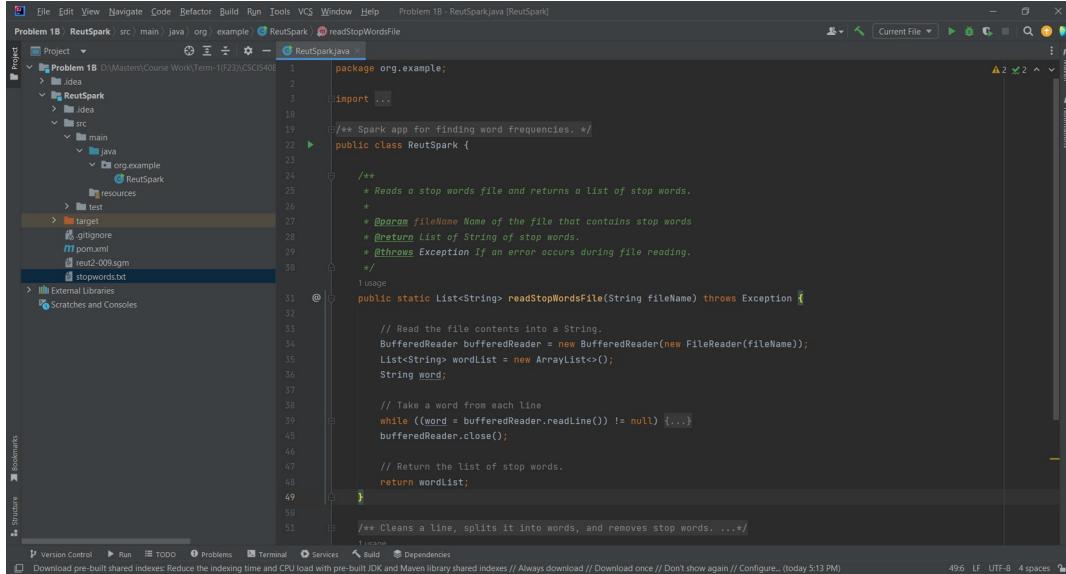
```

Figure 23: stopwords.txt file containing common stop words [8].

Explanation of ReutSpark.java file:

This files contains 3 methods in total, each is explained below:

1. **readStopWordsFile()** – This is just a utility method to read the stopwords.txt file contents and gives back a simple list of string stop-words.



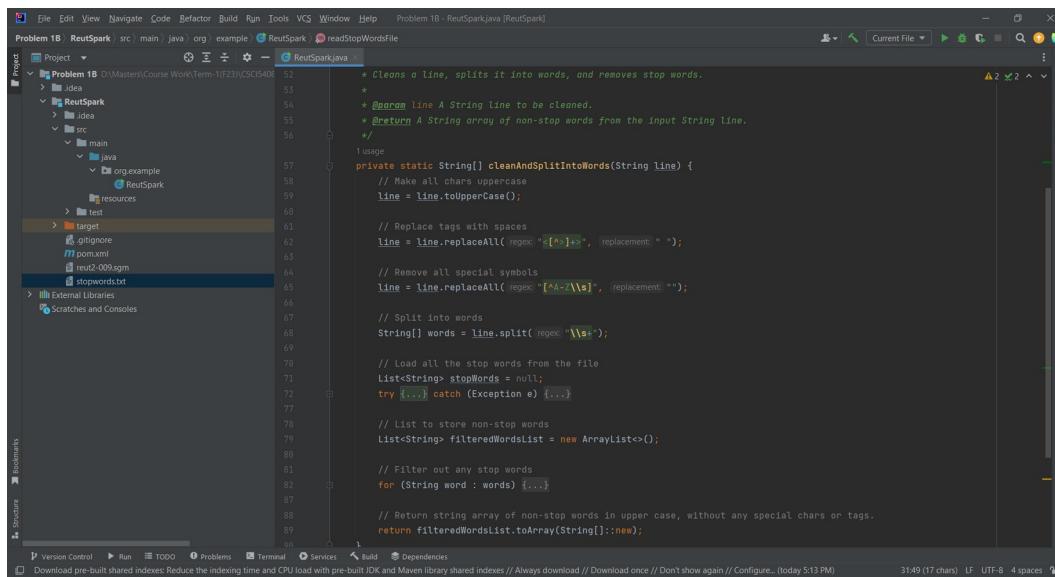
```

1  package org.example;
2
3  import ...
4
5  /**
6   * Spark app for finding word frequencies. */
7  public class ReutSpark {
8
9      /**
10      * Reads a stop words file and returns a list of stop words.
11      *
12      * @param fileName Name of the file that contains stop words.
13      * @return List of String of stop words.
14      * @throws Exception If an error occurs during file reading.
15      */
16     public static List<String> readStopWordsFile(String fileName) throws Exception {
17
18         // Read the file contents into a String.
19         BufferedReader bufferedReader = new BufferedReader(new FileReader(fileName));
20         List<String> wordList = new ArrayList<>();
21         String word;
22
23         // Take a word from each line
24         while ((word = bufferedReader.readLine()) != null) {...}
25         bufferedReader.close();
26
27         // Return the list of stop words.
28         return wordList;
29
30     }
31
32     /**
33      * Cleans a line, splits it into words, and removes stop words. ...*/
34
35     ...
36
37 }

```

Figure 24: readStopWordsFile method logic [2] [3] [4].

2. **cleanAndSplitIntoWords()** – This method first converts all characters in uppercase, so that frequency calculation becomes case insensitive later on. After that, all the tags such as <TITLE>, </TITLE> etc are removed as they are not important for frequency calculation, otherwise such words would be most frequent, just similar to stop words. Then all the special characters are removed and then line is split into words on the basis of space. Then the list of words are filtered on the basis of stopwords.txt file content. The final list of words can now be used further in main method for frequency calculation using spark.



The screenshot shows a Java code editor with the file `ReutSpark.java` open. The code implements the `cleanAndSplitIntoWords` method. The code logic includes:

- Converting the input string to uppercase.
- Replacing tags like <[^>]+> with spaces.
- Replacing all special symbols with an empty string.
- Splitting the cleaned line into words.
- Reading stop words from a file named `stopwords.txt`.
- Filtering out stop words from the word list.
- Returning the array of non-stop words in uppercase.

```

    * Cleans a line, splits it into words, and removes stop words.
    *
    * @param line A String line to be cleaned.
    * @return A String array of non-stop words from the input String line.
    */
    private static String[] cleanAndSplitIntoWords(String line) {
        // Make all chars uppercase
        line = line.toUpperCase();

        // Replace tags with spaces
        line = line.replaceAll("(<[^>]+>)", " ");

        // Remove all special symbols
        line = line.replaceAll("[^A-Z\\s]", "");

        // Split into words
        String[] words = line.split("\\s");

        // Load all the stop words from the file
        List<String> stopWords = null;
        try {
            stopWords = Files.readAllLines(Paths.get("stopwords.txt"));
        } catch (Exception e) {}

        // List to store non-stop words
        List<String> filteredWordsList = new ArrayList<>();

        // Filter out any stop words
        for (String word : words) {
            if (!stopWords.contains(word)) {
                filteredWordsList.add(word);
            }
        }

        // Return string array of non-stop words in upper case, without any special chars or tags.
        return filteredWordsList.toArray(new String[0]);
    }

```

Figure 25: `cleanAndSplitIntoWords` method logic [2] [3] [4] [8].

3. **main()** – The spark session is created first and then we read the .sgm file in terms of spark Dataset of lines. Then we use our utility method `cleanAndSplitIntoWords`, to get the final set of clean words to perform word frequency calculation using spark. Then the `groupByKey` method of spark is used to group the words by their value and get a count for each word. Then all the word frequencies are printed first, followed by highest frequency word and lowest frequency word, with the help of ascending and descending order operations on the count values of these words.

```

public static void main(String[] args) {
    // Create Spark configuration and context
    SparkConf sparkConf = new SparkConf().setAppName("ReutSpark").setMaster("local[*]");
    JavaSparkContext sparkContext = new JavaSparkContext(sparkConf);

    // Create a Spark session
    SparkSession spark = SparkSession.builder().appName("ReutSpark").getOrCreate();

    // Read text file into a Dataset of strings
    Dataset<String> lines = spark.read().textFile(path: "file:///home/adityapurohit27/reut2-009.sgm");

    // Split each line into words
    Dataset<String> words = lines.flatMap((FlatMapFunction<String, String>) line -> Arrays.asList(cleanAndSplitIntoWords(line)).iterator(), Encoders.STRING());

    // Group by word and count frequency.
    Dataset<Row> wordCounts = words.groupBy( col: "value").count();

    // Print all word frequencies.
    System.out.println("All word counts:");
    wordCounts.show(Integer.MAX_VALUE, truncate: false);

    // Get the highest count
    System.out.println("\nHighest Frequency word:");
    System.out.println(wordCounts.orderBy(col( colName: "count").desc()).first());

    // Get the lowest count
    System.out.println("\nLowest Frequency word:");
    System.out.println(wordCounts.orderBy(col( colName: "count").asc(), col( colName: "value").asc()).first());

    // Stop the Spark session and context
    spark.stop();
}

```

Figure 26: main method logic [2] [3] [4] [9] [10].

Functional Testing

Step 1: Create a .jar file using “mvn clean package” command.

```

[INFO] --- surefire:3.1.2:test (default-test) @ ReutSpark ---
[INFO]
[INFO] --- jar:3.0.0:jar (default-jar) @ ReutSpark ---
[INFO] Building jar: D:\Masters\Course Work\Term-1(F23)\CSCI5408-Data\Assignment-2\Problem 1B\ReutSpark\target\ReutSpark-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4.436 s
[INFO] Finished at: 2023-12-01T20:32:06-04:00
[INFO] ------------------------------------------------------------------------
PS D:\Masters\Course Work\Term-1(F23)\CSCI5408-Data\Assignment-2\Problem 1B\ReutSpark>

```

Figure 27: Creating a .jar file

Step-2: Open the ssh of our master node of apache spark cluster and upload 3 files: ReutSpark-1.0-SNAPSHOT.jar, stopwords.txt and reut2-009.sgm

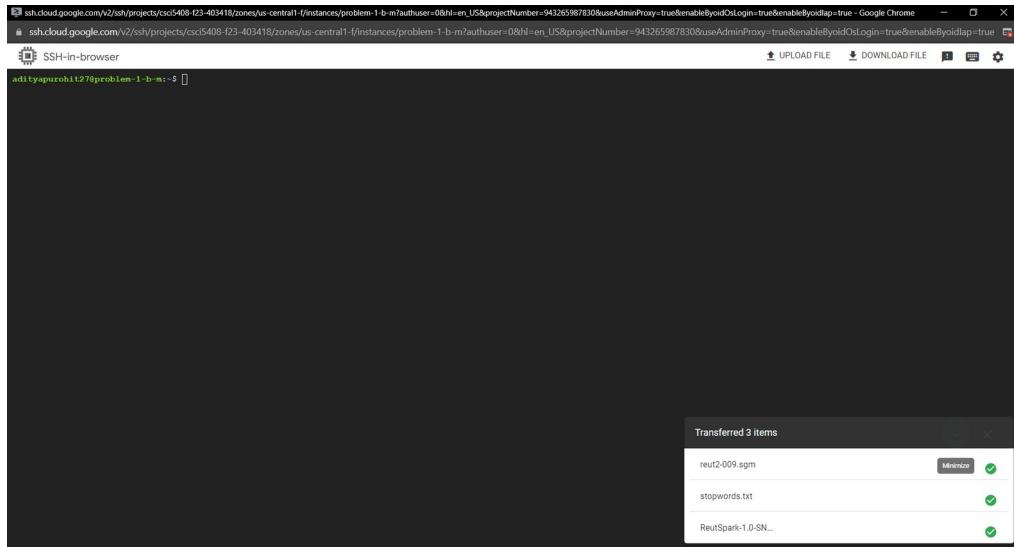


Figure 28: Upload 3 files to the master node [7].

Step-3: Execute the .jar on spark cluster using the command: “spark-submit --class org.example.ReutSpark ReutSpark-1.0-SNAPSHOT.jar”

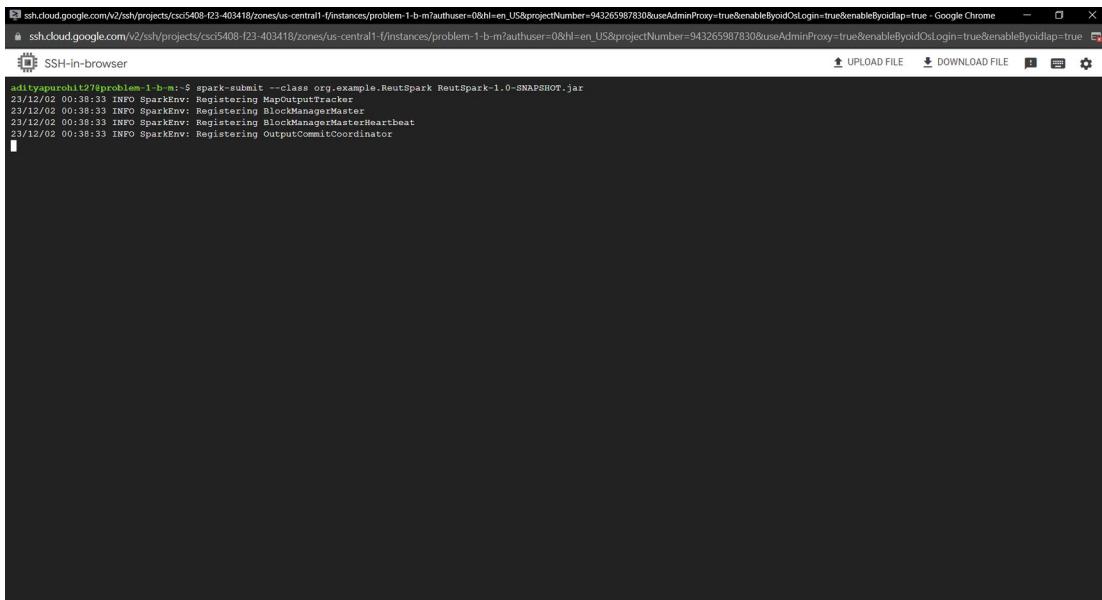


Figure 29: Execute the jar file [7].

```
sh.cloud.google.com/vz/sh/projects/csc5408-f23-403418/regions/us-central1/instances/problem-1-b-m7authuser-08h1-en-US&projectNumber=943265987830&useAdminProxy=true&enableByoidSLogin=true&enableByoidIdp=true - Google Chrome
sh.cloud.google.com/vz/sh/projects/csc5408-f23-403418/regions/us-central1/instances/problem-1-b-m7authuser-08h1-en-US&projectNumber=943265987830&useAdminProxy=true&enableByoidSLogin=true&enableByoidIdp=true
SSH-in-browser
 UPLOAD FILE DOWNLOAD FILE
[<-->]
[<-->]
[BROWSEGERMANCOFFEE] [1] |
[DISMAY] [1] |
[TROLL] [1] |
[LTHRBLZ [1] |
[ATTEMPT] [14] |
[ASSE] [1] |
[ENGAGED] [3] |
[BANG] [3] |
[GEORGIA] [2] |
[KNOCKSARESELLSFOR] [1] |
[LTSHEARSON] [2] |
[HCNTYTRADERSEXPECTCH] [1] |
[HCSHEARERFCNOMUNIT] [1] |
[MCNOMPER] [2] |
[QUICK] [1] |
[TAXATION] [3] |
[HUCKHAM] [1] |
[AMATE] [1] |
[HUSKY] [10] |
[LZCPCIX] [2] |
[ADST] [1] |
[EXECUTION] [3] |
[COMPILED] [1] |
[CHIPMAKERS] [4] |
[HODGINS] [3] |
[DCSDSYNDRREPLACES] [1] |
[WORTHLESS] [1] |
[CALMING] [1] |
[LESTE] [1] |
[DISRUPTIONS] [1] |
[CATALYTIC] [1] |
[WARNERLAMBERTS] [1] |
[<-->]
[<-->]

Highest Frequency word:
[MIN_1284]

Lowest Frequency word:
[ADST]

23/01/02 00:38:57 INFO GfsStorageStatistics: Detected potential high latency for operation op_rename, latencyMs=286; previousMaxLatencyMs=0; operationCount=1; context=rename(gs://dataproc-temp-us-central1-943265987830-izmcq0hd/c27e6b4-c08e-47d7-83e5-d5d8fcf01087695)adictpwurk4t2@problem-1-b-n-3
```

Figure 30: All word counts, with highest and lowest word counts [7].

Expected Output: all word counts with the highest frequency word and lowest frequency word.

Actual Output: As seen in figure 30, all the words and their counts are visible along with the highest frequency and lowest frequency word.

Highest Frequency word:

[MLN, 1284]

Lowest Frequency word:

[AAA, 1]

Flowchart: Spark framework frequency count operation

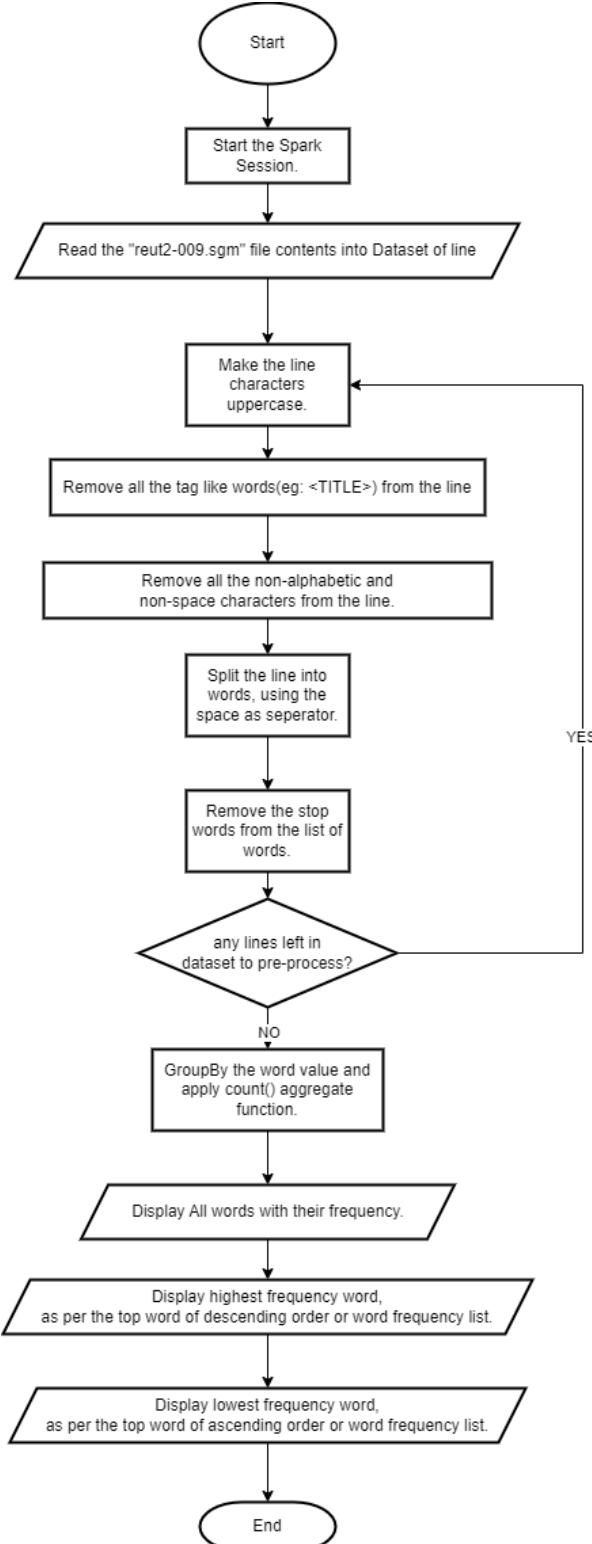


Figure 31: Flowchart of Problem-1B [6].

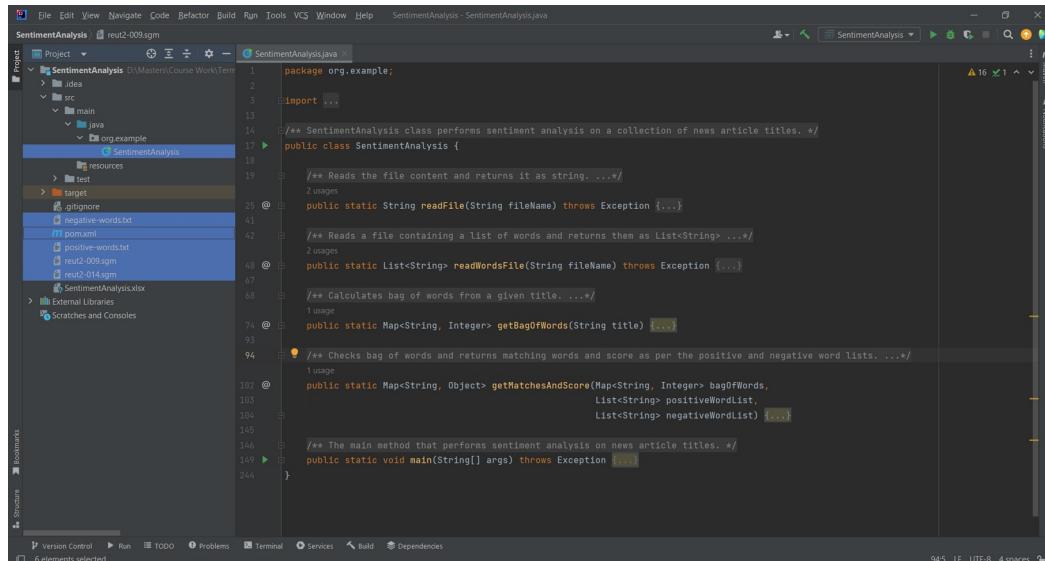
Problem 2: Sentiment Analysis.

Java Program

The java project structure looks something like show in figure 32. There are 6 important files.

SentimentAnalysis.java – This has all the core logic for sentiment analysis of news article titles. All parts of code are commented using Javadoc standards.

reut2-009.sqm and **reut2-014.sqm** are the input files for sentiment analysis.



The screenshot shows a Java development environment with the following details:

- Project Structure:** The project is named "SentimentAnalysis". It contains a "src" directory with "main", "java", and "org.example" packages. "org.example" contains the "SentimentAnalysis" class. Other files in "src" include "resources", "test", "target", "gitignore", and "pom.xml".
- SentimentAnalysis.java Content:**

```
1 package org.example;
2
3 import ...
4
5 /**
6  * SentimentAnalysis class performs sentiment analysis on a collection of news article titles.
7 */
8 public class SentimentAnalysis {
9
10    /**
11     * Reads the file content and returns it as string. ...
12     */
13    public static String readFile(String fileName) throws Exception {...}
14
15    /**
16     * Reads a file containing a list of words and returns them as List<String> ...
17     */
18    public static List<String> readWordsFile(String fileName) throws Exception {...}
19
20    /**
21     * Calculates bag of words From a given title. ...
22     */
23    public static Map<String, Integer> getBagOfWords(String title) {...}
24
25    /**
26     * Checks bag of words and returns matching words and score as per the positive and negative word lists. ...
27     */
28    public static Map<String, Object> getMatchesAndScore(Map<String, Integer> bagOfWords,
29                                                       List<String> positiveWordList,
30                                                       List<String> negativeWordList) {...}
31
32    /**
33     * The main method that performs sentiment analysis on news article titles.
34     */
35    public static void main(String[] args) throws Exception {...}
36}
```
- IDE Interface:** The interface includes tabs for "File", "Edit", "View", "Navigate", "Code", "Refactor", "Build", "Run", "Tools", "VCS", "Window", and "Help". The status bar at the bottom shows "945 LF UTF-8 4 spaces".

Figure 32: Overview of *SentimentAnalysis.java* file and folder structure of project [2] [3] [4]

Pom.xml - 1 apache dependency is used in my maven project's pom.xml in order to export the Sentiment Analysis in an Excel sheet.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <groupId>org.example</groupId>
    <artifactId>SentimentAnalysis</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
    <dependencies>
        <!-- https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml -->
        <dependency>
            <groupId>org.apache.poi</groupId>
            <artifactId>poi-ooxml</artifactId>
            <version>5.2.5</version>
        </dependency>
    </dependencies>
</project>

```

Figure 33: pom.xml file of Problem-2 [11].

positive-words.txt and **negative-words.txt**– These two files contain positive and negative word list. These are downloaded from GitHub [12] [13].

positive-words.txt	negative-words.txt
;	1. The appearance of an opinion word in a sentence.
;	mean that the sentence expresses a positive or negative
;	See the paper below:
;	Bing Liu. "Sentiment Analysis and Subjectivity." An chapter
;	Handbook of Natural Language Processing, Second Edition
;	(editors: N. Indurkha and F. J. Damerau), 2010.
;	;
;	2. You will notice many misspelled words in the list. They
;	mistakes. They are included as these misspelled words appear
;	frequently in social media content.
a+	;
abound	2-faced
abounds	2-faces
abundance	abnormal
abundant	abominable
accessible	abominably
accessible	abominate
acclaim	abomination
acclaimed	abort
acclamation	aborted
accolade	aborts
accolades	abrade
accommodative	abrasive
accommodative	abrupt
accomplish	abruptly
accomplished	abscond
accomplishment	absence
accomplishments	absent-minded

Figure 34: Positive and Negative word list [12] [13].

Explanation of SentimentAnalysis.java file:

This files contains 5 methods in total, each is explained below:

1. **readWordsFile()** – This is just a utility method to read the positive-words.txt and negative-words.txt file contents and gives back a simple list of words.

The screenshot shows the Java code for the `readWordsFile` method in the `SentimentAnalysis` class. The code reads two files: `positive-words.txt` and `negative-words.txt`. It uses a `BufferedReader` to read lines from the files and adds them to a `List<String>`. The code includes comments explaining the purpose of each section and the use of the `BufferedReader` class.

```
19     /** Reads the file content and returns it as string. ....*/
20     2 usages
21     public static String readFile(String fileName) throws Exception {
22
23         // Read the file contents into a String.
24         BufferedReader bufferedReader = new BufferedReader(new FileReader(fileName));
25         List<String> wordList = new ArrayList<>();
26
27         String word;
28
29         // Keep reading until the end of file
30         while ((word = bufferedReader.readLine()) != null) {
31             if (word.startsWith("//") || word.isEmpty()) {
32                 continue;
33             }
34             wordList.add(word);
35         }
36
37         bufferedReader.close();
38
39         // Return the List<String>
40         return wordList;
41     }
42
43     /** Calculates bag of words from a given title. ....*/
44     1 usage
45     public static Map<String, Integer> getBagOfWords(String title) { ... }
```

Figure 35: Logic of readWordsFile method [2] [3] [4].

2. **readFile()** – This is also a simple utility method to read the content of the .sgm files using the BufferedReader Class of java.

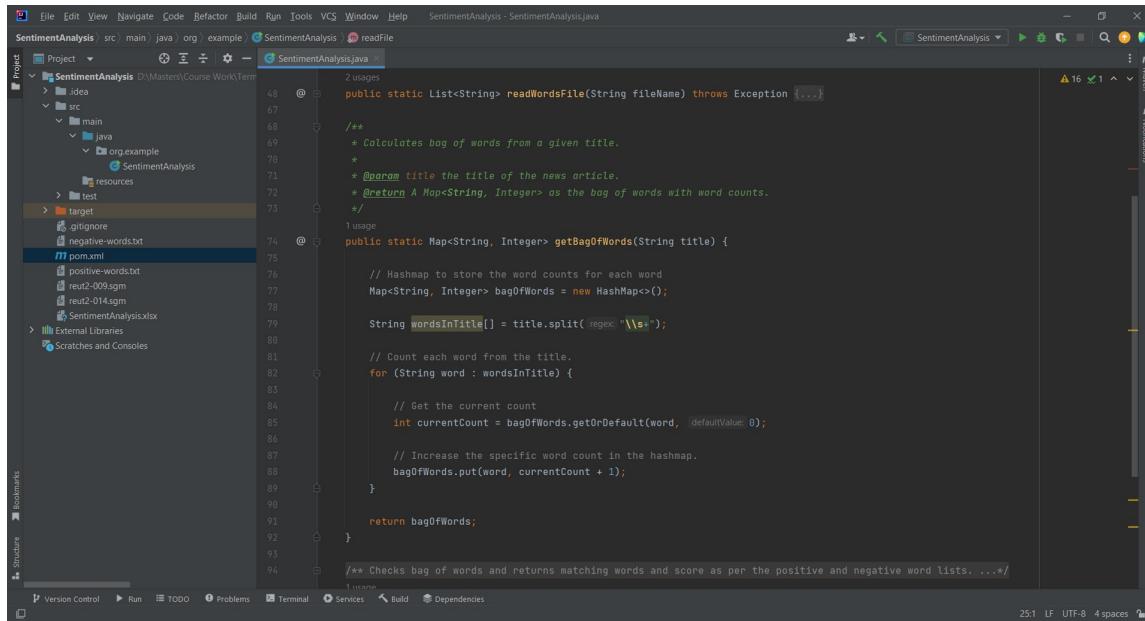
The screenshot shows the Java code for the `readFile` method in the `SentimentAnalysis` class. The code reads a file named `title.sgm` and returns its content as a string. It uses a `BufferedReader` to read lines from the file and appends them to a `StringBuilder`. The code includes comments explaining the purpose of each section and the use of the `BufferedReader` class.

```
18     /**
19      * Reads the file content and returns it as string.
20      *
21      * @param fileName Name of file to read.
22      * @return File contents as string
23      */
24     2 usages
25     public static String readFile(String fileName) throws Exception {
26
27         // Read the file contents into a String.
28         BufferedReader bufferedReader = new BufferedReader(new FileReader(fileName));
29         StringBuilder sgmFileContentTemp = new StringBuilder();
30
31         // Keep reading until the end of file
32         String lineOfFile;
33         while ((lineOfFile = bufferedReader.readLine()) != null) {
34             sgmFileContentTemp.append(lineOfFile);
35         }
36
37         bufferedReader.close();
38
39         // Return the file contents as a string.
40         return sgmFileContentTemp.toString();
41     }
42
43     /** Returns a file containing a list of words and returns them as List<String>....*/
44     1 usage
45     public static List<String> readWordFile(String fileName) throws Exception { ... }
```

Figure 36: readFile method logic [2] [3] [4] [8].

3. **getBagOfWords()** – This method is used to create a `HashMap` of words and their respective counts in the title. First of all, an empty Map is created and then the title is split

into an array of strings by the help of space separator. Then for each word, we get the current count of it in the HashMap if already exists and add a 1 to it, else we just set it 1 if it is being counted for the first time in the title. After the array is processed fully, the final HashMap is returned.



The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The project is named "SentimentAnalysis". It contains a "src" folder with "main" and "test" subfolders. "main" contains "java" and "org.example" packages, with "SentimentAnalysis.java" being the active file. "test" contains "resources" and "target" folders. "resources" contains "positive-words.txt", "negative-words.txt", and "pom.xml". "target" contains "positive-words.txt", "read2-009.qm", "read2-014.qm", and "SentimentAnalysis.xlsx".
- SentimentAnalysis.java:** The code implements the `getBagOfWords` method. It reads words from a file, splits them into an array, and then iterates through each word to update its count in a HashMap. If the word is present in the HashMap, its count is increased by 1. Otherwise, it is added with a count of 1. Finally, the method returns the HashMap.
- Code Snippet:**

```

    public static Map<String, Integer> getBagOfWords(String title) {
        Map<String, Integer> bagOfWords = new HashMap<>();
        String[] wordsInTitle = title.split("\\s+");
        for (String word : wordsInTitle) {
            int currentCount = bagOfWords.getOrDefault(word, 0);
            bagOfWords.put(word, currentCount + 1);
        }
        return bagOfWords;
    }

```

Figure 37: `getBagOfWords` method logic [2] [3] [4].

4. **getMatchesAndScore()** – This method is used to calculate the matching words part of sentiment analysis and the score which can be -ve, 0 or +ve. I have first set the score to 0 and matching words as empty. Then for each word in the bagOfWords, I check it with the positive word list. If present, I add +1 to score and that word into matches list. If not, I check it with the negative word list; if a match occurs here, -1 is done in the score and word is added into matches list. But if the word is not found in any of the 2 lists, nothing is added to the score and nothing is added to the matches list for that word. In this way, whole bagOfWord is processed to get the sentiment match column and score column value.

```

/*
 * Checks bag of words and returns matching words and score as per the positive and negative word Lists.
 *
 * @param bagOfWords The bag of words to check.
 * @param positiveWordList List of positive words.
 * @param negativeWordList List of negative words.
 * @return A Map<String, Object> containing matches and overall sentiment score.
 */
1 usage
public static Map<String, Object> getMatchesAndScore(Map<String, Integer> bagOfWords,
                                                       List<String> positiveWordList,
                                                       List<String> negativeWordList) {
    // Create a HashMap to store matches and score
    Map<String, Object> output = new HashMap<>();

    List<String> matches = new ArrayList<>();
    int overallScore = 0;

    outerLoop:
    for (String word : bagOfWords.keySet()) {

        // Check if current word is positive.
        for (String positiveWord : positiveWordList) {
            if (positiveWord.toUpperCase().equals(word.toUpperCase())) {
                // Add a match
                matches.add(word);
                // Add count of that word into overall score.
                overallScore += bagOfWords.get(word);

                // If the word is positive, no need to check negative list. Just start matching process for next word.
                continue outerLoop;
            }
        }
    }
}

```

Figure 38: *getMatchesAndScore* method logic [2] [3] [4].

5. **main()** – In this method, both the .sgm files are first read. The second file is appended with the first for easier processing later on. Then the lists of positive and negative words are populated from the .txt files. Then the regex is prepared for finding the content between titles. This regex is the subpart of the regex which was used in Problem 1A. Then an empty Excel sheet is created with just header row as of now. As per the regex matcher, all the titles are iterated from the top to bottom of the file contents and for each of the title, sentiment analysis is done and stored as a new row in excel sheet. Before finding bagOfWords, each title is made uppercase to make it case insensitive. Some special characters like comma, > and < are also removed for cleaning the title.

The match and score values are generated with the help of *getMatchesAndScore* method as explained earlier. The polarity is determined as positive for +ve score, negative for -ve score value and else neutral. Along with adding analysis data to excel, it will be also printed on console. At last, the excel columns are resized and file is saved as *SentimentAnalysis.xlsx*

```

169 	/** The main method that performs sentiment analysis on news article titles. */
170
171 	public static void main(String[] args) throws Exception {
172
173 		// Read the content of both the files.
174 		String fileContent = "";
175
176 		try {
177 			fileContent = readFile("filename:./reut2-009.sgm");
178 			fileContent += fileContent + readfile("filename:./reut2-014.sgm");
179
180
181 		} catch (Exception e) {
182 			System.out.println(e);
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330

```

Figure 39: main() method logic [2] [3] [4].

Functional Testing

Just execute the main method of the java program. Make sure both .sgm files are in the root directory of the project.

Expected Output: Console should print the sentiment analysis for each news title and lastly create an excel sheet for the same.

Actual Output: Excel is generated along with console output of each title's sentiment analysis.

```

news1693 = "EQUITABLE RESOURCES EQT FILES UNIT OFFERING"
bonds1693 = [RESOURCES=1, OFFERING=1, UNIT=1, EQT=1, FILES=1, EQUITABLE=1]
Matches1693 = EQUITABLE
Score1693 = 1
Polarity1693 = Positive

news1694 = "TOWN AND COUNTRY JEWELRY MANUFACTURING TOJC"
bonds1694 = [COUNTRY=1, TOWN=1, AND=1, MANUFACTURING=1, JEWELRY=1, TOJC=1]
Matches1694 = 0
Score1694 = Neutral
Polarity1694 = Neutral

Sentiment Analysis Report is generated in file: ./SentimentAnalysis.xlsx

```

Figure 40: Execution of Sentiment Analysis program [2] [3] [4].

A	B	C	D	E	F	G	H	I	J	K	L	M
News#	File	match	score	Polarity								
1	ADVANCED MAGNETICS ADMG IN AGREEMENT	ADVANCED	1	Positive								
2	HEALTH RESEARCH FILES FOR BANKRUPTCY		0	Neutral								
3	3 NUMERIX CORP NMRX 2ND QTR JAN 31 LOSS	LOSS	-1	Negative								
4	4 U.S. SELLING 12.8 BILLION DLS OF 3 AND 6-MO BILLS MARCH 30 TO PAY DOWN 1.2 BILLION DLRs		0	Neutral								
5	5 U.S. 2-YEAR NOTE AVERAGE YIELD 6.43 PCT STOP 6.44 PCT AWARDED AT HIGH RYD 85 PCT	AWARDED	1	Positive								
6	6 COMMODORE CRU ATARI IN SETTLEMENT		0	Neutral								
7	7 BALDRIGE SUPPORTS NIC TALKS ON CURRENCIES	SUPPORTS	1	Positive								
8	8 TRIANGLE TRI BEGINS EXCHANGE OFFER		0	Neutral								
9	9 SOUTHMARK SM UNIT IN PUBLIC OFFERING OF STOCK		0	Neutral								
10	10 EASTMAN KODAK CO TO SELL HOLDINGS IN ICN PHARMACEUTICALS AND VIRATEK INC		0	Neutral								
11	11 FEUD PERSISTS AT U.S. HOUSE BUDGET COMMITTEE		0	Neutral								
12	12 TREASURY BALANCES AT FED ROSE ON MARCH 23		0	Neutral								
13	13 FARM CREDIT SYSTEM SEEN NEEDING 800 MLN DLRs AID		0	Neutral								
14	14 USX X USS UNI RAISES PRICES		0	Neutral								
15	15 UNIONIST URGES RETALIATION AGAINST JAPAN		0	Neutral								
16	16 EXXON (XON) GETS 99.2 MLN DLR CONTRACT		0	Neutral								
17	17 ZAMBIA AUTHORIZES TO BUY PL-480 RICE - USDA		0	Neutral								
18	18 ZAIRE AUTHORIZED TO BUY PL-480 RICE - USDA		0	Neutral								
19	19 MCDONNELL DOUGLAS GETS 30.6 MLN DLR CONTRACT		0	Neutral								
20	20 MIDWEST ACQUIRES ASSETS OF BUSINESS AVIATION		0	Neutral								
21	21											

Figure 41: Excel file of Sentiment analysis [11] [14].

References:

- [1] “Download MongoDB Community Server,” *MongoDB*. [Online]. Available: <https://www.mongodb.com/try/download/community>. [Accessed: 24-Nov-2023].
- [2] “Java | Oracle,” *Java.com*. [Online]. Available: <https://www.java.com/en/>. [Accessed: 25-Nov-2023].
- [3] “IntelliJ IDEA – the leading Java and Kotlin IDE,” *JetBrains*. [Online]. Available: <https://www.jetbrains.com/idea/>. [Accessed: 25-Nov-2023].
- [4] B. Porter, J. van Zyl, and O. Lamy, “Welcome to Apache maven,” *Apache.org*. [Online]. Available: <https://maven.apache.org/>. [Accessed: 25-Nov-2023].
- [5] “Maven Repository: org.mongodb» mongo-java-driver» 3.12.14,” *Mvnrepository.com*. [Online]. Available: <https://mvnrepository.com/artifact/org.mongodb/mongo-java-driver/3.12.14>. [Accessed: 25-Nov-2023].
- [6] “Flowchart maker & online diagram software,” *Diagrams.net*. [Online]. Available: <https://app.diagrams.net/>. [Accessed: 30-Nov-2023].
- [7] “Dataproc,” *Google Cloud*. [Online]. Available: <https://cloud.google.com/dataproc?hl=en>. [Accessed: 01-Dec-2023].
- [8] larsyencken, “Stopwords.Txt,” *GitHub*. [Online]. Available: <https://gist.github.com/larsyencken/1440509>. [Accessed: 01-Dec-2023].
- [9] “spark-core,” *Mvnrepository.com*. [Online]. Available: <https://mvnrepository.com/artifact/org.apache.spark/spark-core>. [Accessed: 01-Dec-2023].
- [10] “spark-sql,” *Mvnrepository.com*. [Online]. Available: <https://mvnrepository.com/artifact/org.apache.spark/spark-sql>. [Accessed: 01-Dec-2023].
- [11] “Apache poi-ooxml,” *Mvnrepository.com*. [Online]. Available: <https://mvnrepository.com/artifact/org.apache.poi/poi-ooxml/5.2.5>. [Accessed: 01-Dec-2023].
- [12] Marcin, “positive-words.txt,” *GitHub*. [Online]. Available: <https://gist.github.com/mkulakowski2/4289437>. [Accessed: 01-Dec-2023].
- [13] Marcin, “negative-words.txt,” *GitHub*. [Online]. Available: <https://gist.github.com/mkulakowski2/4289441>. [Accessed: 01-Dec-2023].

- [14] “Microsoft Excel,” *Microsoft.com*. [Online]. Available: <https://www.microsoft.com/en-ca/microsoft-365/excel>. [Accessed: 01-Dec-2023].