Module 2 Programing Assignment

Max Robinson

1 Summary

In this assignment a document set, "Headlines" was processed to produce a summary of the document set, and inverted file written to disk as a binary file, and a lexicon written to disk as text.

The document sets were contained in single text files where each document in the file was identified by surrounding text marking the stop and start of the document, and providing the document ID.

When processing the documents the text was broken into words and normalized. Words were determined by splitting the text on spaces. Internal punctuation was left intact. The words were then normalized by lower-casing all words. Any word that was itself only punctuation was also disregarded. The set of characters characterized as punctuation are as follows: '.', ',', '"', '"', '?', '!', ':', ';', '(', ')', '[', ']', '', '', '&', '—', '-'.

As the terms were read, they were added to the lexicon and the postings list for that term. The lexicon was structured as a HashMap of terms to LexiconItems. Lexicon items are objects that contain the term id, document frequency, and offset of the term in the inverted file.

While constructing the lexicon, an in memory map of terms to postings list was constructed as well. This mapped terms to posting lists. Each posting holds the document id and the term frequency for that document.

The inverted file was then written out to disk after all documents had been processed. The inverted file was written out by first sorting all of the terms in lexicographical order, and then writing the terms corresponding postings list out to a binary file using 4-byte integers. The integers were written in order of document id, then term frequency, using a total of 8 bytes per posting.

Before each postings list was written out, the offset for the start of the postings list is recorded in the lexicon so that each term knows where the start of their postings list is.

The lexicon was then written out as a text file of symbol delineated values. To avoid running into other likely symbols in the documents, combination symbol of "{&" was used to delimit the values of *term*, *termID*, *document frequency*, and *offset*.

To find values after the lexicon and inverted files had been created, the lexicon was read back into memory, and a search was conducted based on the terms existence in the lexicon. If a term was found in the lexicon, the lexicon

then read from the inverted file on disk by seeking to the offset and then reading in the appropriate number of documents, according to the document frequency stored in the lexicon. For each document, 8 bytes were read from the inverted file to recover the document id and the term frequency.

Three tests were then performed to check for correctness of the program. The results are summarized in section 4.

2 Program Summary Results

The following are summary results from running the main ingest program "Module2.java" which ingests all the documents, creates the lexicon and inverted files, and writes them out to disk.

```
Number of Documents: 500000
Number of unique terms: 262644
Number of terms observed: 4548579
```

3 File Summary

The size of the inverted file on disk was: 35753944 bytes
The size of the lexicon file on disk was: 7611091 bytes
Together the total size was: 43365035 bytes
The original file size was: 39381610 bytes

In this instance the inverted index and lexicon together took up more space than the original text file. I believe that this is the case because, unlike in the first assignment, punctuation was not removed from the start and end of terms this time when indexing. As a result, I believe that many terms had punctuation attached to them, and as a result took up more space.

Overall, the inverted file took up more space than the lexicon / dictionary.

4 Tests

4.1 Test 1

Print out the document frequency and postings list for terms: Heidelberg", "plutonium", Omarosa, octopus".

```
Term: heidelberg Document Freq: 8
Postings: [(DocID=114330, DocCount=1), (DocID=135134, DocCount=1),
 (DocID=174781, DocCount=1), (DocID=221100, DocCount=1), (DocID=243838,
 DocCount=1), (DocID=452546, DocCount=1), (DocID=491140, DocCount=1),
 (DocID=491279, DocCount=1)]

Term: plutonium Document Freq: 5
```

```
Postings: [(DocID=63765, DocCount=1), (DocID=202980, DocCount=1),
 (DocID=297335, DocCount=1), (DocID=314543, DocCount=1), (DocID=435698,
 DocCount=1)]

Term: omarosa Not found
Postings: None

Term: octopus Document Freq: 16
Postings: [(DocID=27847, DocCount=1), (DocID=102608, DocCount=1),
 (DocID=164437, DocCount=1), (DocID=168891, DocCount=1), (DocID=171010,
 DocCount=1), (DocID=176081, DocCount=1), (DocID=184660, DocCount=1),
 (DocID=188627, DocCount=1), (DocID=273038, DocCount=1), (DocID=318710,
 DocCount=1), (DocID=350246, DocCount=1), (DocID=377653, DocCount=1),
 (DocID=414160, DocCount=1), (DocID=444427, DocCount=1), (DocID=448376,
 DocCount=1), (DocID=474571, DocCount=1)]
```

4.2 Test 2

Give document frequency, but do not print postings for the words Hopkins", Harvard, Stanford, and, college (these postings lists are longer).

```
Term: hopkins Document Freq: 55
Term: harvard Document Freq: 79
Term: stanford Document Freq: 127
Term: college Document Freq: 1845
```

4.3 Test 3

Print out the docids that have both "Jeff" and "Bezos" in the text. You can do this by finding the postings lists for each term, and then intersecting the two lists. For this test you do not need to use frequency information. Please print the docids in increasing order.

```
[11767, 25061, 57126, 172835, 261283, 263522, 270753, 275521, 290723, 314604,
 321472, 371274, 381540, 449685]
```

5 Code

By Max Robinson
The following subsections provide the code that was created to process and produce the results for this assignment.

5.1 Module2.java - Max Robinson

The main program for ingesting the documents, building the lexicon and inverted file, and writing it out to disk.

```
package edu.jhu.mrobi100;


public class Module2 {
  private static String invertedFileName =
 "/home/max/Documents/JHU-Masters/InformationRetrieval" +
          "/Module2/module2/src/main/java/edu/jhu/mrobi100/invertedFile.txt";
  private static String lexiconFileName =
 "/home/max/Documents/JHU-Masters/InformationRetrieval" +
          "/Module2/module2/src/main/java/edu/jhu/mrobi100/lexicon.txt";


  public static void main(String[] args) throws Exception {
    // Create Lexicon and Inverted File
    Ingest ingest = new Ingest(Module2.class.getResource("headlines.txt")
        .toURI().toString().substring(5));
    Lexicon lexicon = ingest.read();

    lexicon.writeInvertedFile(invertedFileName);
    lexicon.writeLexicon(lexiconFileName);

    System.out.println("Number of Documents: " +
             ingest.getNumberOfDocuments());
    System.out.println("Number of unique terms: " +
             lexicon.getDict().keySet().size());
    System.out.println("Number of terms observed: " +
             ingest.getNumberOfTerms());
  }
}
```

5.2 Tests.java - Max Robinson

The main program that used the on disk lexicon and inverted file to answer questions about the data.

```
package edu.jhu.mrobi100;

import java.net.BindException;
import java.util.*;

public class Tests {
  private static String invertedFileName =
 "/home/max/Documents/JHU-Masters/InformationRetrieval" +
          "/Module2/module2/src/main/java/edu/jhu/mrobi100/invertedFile.txt";
  private static String lexiconFileName =
```

```
"/home/max/Documents/JHU-Masters/InformationRetrieval" +
        "/Module2/module2/src/main/java/edu/jhu/mrobi100/lexicon.txt";

 public static void main(String[] args) throws Exception {
   System.out.println("TEST 1");
   test1();
   System.out.println("TEST 2");
   test2();
   System.out.println("TEST 3");
   test3();
 }

 public static void test1() throws Exception {
   // Read Lexicon from Disk
   Lexicon lexicon2 = new Lexicon();
   lexicon2.readLexicon(lexiconFileName);
   String[] result = null;
   result = lexicon2.find("Heidelberg", invertedFileName);
   System.out.println(result[0]);
   System.out.println(result[1]);
   result = lexicon2.find("plutonium", invertedFileName);
   System.out.println(result[0]);
   System.out.println(result[1]);
   result = lexicon2.find("Omarosa", invertedFileName);
   System.out.println(result[0]);
   System.out.println(result[1]);
   result = lexicon2.find("octopus", invertedFileName);
   System.out.println(result[0]);
   System.out.println(result[1]);
 }

 public static void test2() throws Exception {
   // Read Lexicon from Disk
   Lexicon lexicon2 = new Lexicon();
   lexicon2.readLexicon(lexiconFileName);
   String[] result = null;
   result = lexicon2.find("Hopkins", invertedFileName);
   System.out.println(result[0]);
   result = lexicon2.find("Harvard", invertedFileName);
   System.out.println(result[0]);
   result = lexicon2.find("Stanford", invertedFileName);
   System.out.println(result[0]);
   result = lexicon2.find("college", invertedFileName);
   System.out.println(result[0]);
 }
```

```
  public static void test3() throws Exception {
    // Read Lexicon from Disk
    Lexicon lexicon2 = new Lexicon();
    lexicon2.readLexicon(lexiconFileName);

    List<Integer> jeff_res= lexicon2.findDocIDs("Jeff", invertedFileName);
    List<Integer> Bezos_res= lexicon2.findDocIDs("Bezos", invertedFileName);

    Set<Integer> jeff = new HashSet<>(jeff_res);
    Set<Integer> bezos = new HashSet<>(Bezos_res);

    // Now only keep elements in common
    jeff.retainAll(bezos);

    ArrayList<Integer> results = new ArrayList<>(jeff);
    Collections.sort(results);

    System.out.println(results);
  }
}
```

5.3 Ingest.java - Max Robinson

The file responsible for data ingest and term normalization.

```
package edu.jhu.mrobi100;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.*;
import java.util.regex.Pattern;

public class Ingest {
  private final String fileName;
  private final String startParagraph = "<P ID=";
  private final Pattern start = Pattern.compile(startParagraph);
  private final Pattern stop = Pattern.compile("</P>");
  private final List<String> punct =
      Arrays.asList(".", ",", "'", "?", "!", ":", ";", "(", ")", "[", "]",
 "{", "}", "&", "|", "-");

  private final Postings postings;
  private final Lexicon lexicon;

  private int numberOfDocuments = 0;
```

```
private int numberOfTerms = 0;

public Ingest(String fileName) throws Exception {
  this.fileName = fileName;
  this.postings = new Postings();
  this.lexicon = new Lexicon();
}

public Lexicon read() throws IOException {
  try (BufferedReader bf = new BufferedReader(new FileReader(fileName))) {
    String line;
    StringBuilder document = new StringBuilder();
    int id = -1;
    while ((line = bf.readLine()) != null) {

      line = line.trim();

      if (start.matcher(line).find()) {
        id = parseStart(line);
      } else if (stop.matcher(line).find()) {
        updateTerms(document.toString(), id);
        document.delete(0, document.length());
        id = -1;
        numberOfDocuments+=1;
      } else {
        document.append(line);
      }
    }

  } catch (Exception ex) {
    throw ex;
  }

  return lexicon;
}

private void updateTerms(String line, int id) {
  HashMap<String, Integer> termFreq = new HashMap<>();

  String[] tokens = line.split(" ");
  for (String token : tokens) {
    token = token.toLowerCase();
    if (punct.contains(token)) {
      continue;
    }
```

```
      numberOfTerms+=1;
      Integer df = termFreq.get(token);
      if(df == null){
        termFreq.put(token, 1);
      } else {
        termFreq.put(token, df + 1);
      }
    }

    for(Map.Entry<String, Integer> entry: termFreq.entrySet()){
      String key = entry.getKey();
      Integer df = entry.getValue();
      lexicon.addTerm(key, id, df);
    }
  }

  private int parseStart(String line) {
    line = line.replace("<P ID=", "");
    line = line.replace(">", "");
    int id = -1;
    try {
      id = Integer.parseInt(line);
    } catch (Exception ex) {
    }
    return id;
  }

  // <editor-fold desc="Accessors">

  public int getNumberOfDocuments() {
    return numberOfDocuments;
  }

  public int getNumberOfTerms() {
    return numberOfTerms;
  }

  // </editor-fold>
}
```

5.4 Lexicon.java - Max Robinson

The work horse of the program. The lexicon holds a dictionary of terms to LexiconItems, and a Postings data structure. The lexicon class is responsible for all of the writing to and from files, as well as searching for data once everything has been indexed. The use of the Postings data structure in the class is a

convenient way to consolidate data during data ingestion so that writing out to an inverted file is easy and abstracted from the main program.

```
package edu.jhu.mrobi100;

import java.io.*;
import java.util.*;

public class Lexicon {
  private static int TermID = 0;

  private HashMap<String, LexiconItem> dict;
  private Postings postings;

  public Lexicon() {
    dict = new HashMap<>();
    postings = new Postings();
  }

  public void addTerm(String term, int docID, Integer termFrequency) {

    // Add the term to the lexicon
    LexiconItem item = dict.get(term);
    if (item == null) {
      LexiconItem entry = new LexiconItem(TermID);
      entry.setDf(1);
      dict.put(term, entry);
      TermID += 1;
    } else {
      item.addDf(1);
    }

    // add the term to the postings list
    Posting posting = new Posting(docID, termFrequency);
    ArrayList<Posting> postings = this.postings.getDict().get(term);
    if (postings == null) {
      ArrayList<Posting> newPostings = new ArrayList<>();
      newPostings.add(posting);
      this.postings.getDict().put(term, newPostings);
    } else {
      postings.add(posting);
      this.postings.getDict().put(term, postings);
    }
  }

  public void writeLexicon(String fileName){
```

```
  try(BufferedWriter bw = new BufferedWriter(new FileWriter(fileName))){
    Set<String> keySet = dict.keySet();
    ArrayList<String> keys = new ArrayList<>(keySet);
    Collections.sort(keys);

    for(String key : keys){
      LexiconItem item = dict.get(key);
      String output = key +"{&" + item.getId() + "{&" + item.getDf() + "{&"
                          + item.getOffset()+"\n";
      bw.write(output);
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}

public void readLexicon(String fileName) throws Exception{
  try (BufferedReader bf = new BufferedReader(new FileReader(fileName))) {
    String line;
    while ((line = bf.readLine()) != null) {
      line = line.trim();
      String[] parts = line.split("\\{&");

      String term = parts[0];
      try{
        int id = Integer.parseInt(parts[1]);
        int df = Integer.parseInt(parts[2]);
        int offset = Integer.parseInt(parts[3]);

        LexiconItem item = new LexiconItem(id);
        item.setDf(df);
        item.setOffset(offset);

        this.dict.put(term, item);
      }catch (NumberFormatException ex){
        System.out.println(line);
      }

    }

  } catch (Exception ex) {
    throw ex;
  }
}

public void writeInvertedFile(String fileName) throws IOException{
```

```
  RandomAccessFile rfa = new RandomAccessFile(fileName, "rw");

  Set<String> keySet = dict.keySet();
  ArrayList<String> keys = new ArrayList<>(keySet);
  Collections.sort(keys);

  int offset = 0;
  for(String key : keys){
    LexiconItem item = dict.get(key);
    item.setOffset((int)rfa.getFilePointer());
    offset += writePostings(rfa, postings.getDict().get(key));
  }
}

private int writePostings(RandomAccessFile rfa,  ArrayList<Posting> postings)
                          throws IOException {
  int offset = 0;
  for(Posting posting : postings){
    rfa.writeInt(posting.getDocID());
    rfa.writeInt(posting.getDocCount());
    offset += 8;
  }

  return offset;
}

// <editor-fold desc="Accessors">
public HashMap<String, LexiconItem> getDict() {
  return dict;
}

public Postings getPostings() {
  return postings;
}
// </editor-fold>

public String[] find(String term, String fileName) {
  term = term.trim();
  term = term.toLowerCase();
  LexiconItem li = dict.get(term);
  if(li == null){
    return new String[]{"Term: "+ term + " Not found", "Postings: None"};
  }
  ArrayList<Posting> postings = getPostingsFromDisk(li, fileName);
  return new String[] {"Term: " + term + " Document Freq: " + li.getDf(),
           "Postings: " + postings};
```

```
  }

  public List<Integer> findDocIDs(String term, String fileName) {
    term = term.trim();
    term = term.toLowerCase();
    LexiconItem li = dict.get(term);
    if(li==null){
      return Collections.emptyList();
    }
    ArrayList<Posting> postings = getPostingsFromDisk(li, fileName);
    ArrayList<Integer> results = new ArrayList<>();
    for(Posting p : postings){
      results.add(p.getDocID());
    }
    return results;
  }

  private ArrayList<Posting> getPostingsFromDisk(LexiconItem li, String fileName) {
    ArrayList<Posting> postings = new ArrayList<>();
    try(RandomAccessFile rfa = new RandomAccessFile(fileName, "rw")){
      rfa.seek(li.getOffset());

      for (int i = 0; i < li.getDf(); i++) {
        int docID = rfa.readInt();
        int docCount = rfa.readInt();

        Posting p = new Posting(docID, docCount);
        postings.add(p);
      }
    } catch (FileNotFoundException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace();
    }
    return postings;
  }


  @Override
  public String toString() {
    return "Lexicon{" + "dict=" + dict + ", postings=" + postings + '}';
  }
}
```

5.5 LexiconItem.java - Max Robinson

Data structure for holding lexicon info.

```
package edu.jhu.mrobi100;

public class LexiconItem {
  private int id;
  private int df;
  private int offset;

  public LexiconItem(int id) {
    this.id = id;
    df = 0;
    offset = 0;
  }

  public void addDf(int df){
    this.df += df;
  }

  // <editor-fold desc="Accessors">
  public int getId() {
    return id;
  }

  public void setId(int id) {
    this.id = id;
  }

  public int getDf() {
    return df;
  }

  public void setDf(int df) {
    this.df = df;
  }

  public int getOffset() {
    return offset;
  }

  public void setOffset(int offset) {
    this.offset = offset;
  }
  // </editor-fold>
```

```
  @Override
  public String toString() {
    return "LexiconItem{" +
            "id=" + id +
            ", df=" + df +
            ", offset=" + offset +
            '}';
  }
}
```

5.6 Postings.java - Max Robinson

Support data structure for keeping a map of terms to postings list.

```
package edu.jhu.mrobi100;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

public class Postings {
  private HashMap<String, ArrayList<Posting>> dict;

  public Postings() {
    dict = new HashMap<>();
  }

  public HashMap<String, ArrayList<Posting>> getDict() {
    return dict;
  }

  @Override
  public String toString() {
    return "Postings{" +
            "dict=" + dict +
            '}';
  }
}
```

5.7 Posting.java - Max Robinson

Data structure for holding posting information.

```
package edu.jhu.mrobi100;

public class Posting {
```

```
  private int DocID;
  private int DocCount;

  public Posting(int docID, int docCount) {
    DocID = docID;
    DocCount = docCount;
  }

  public int getDocID() {
    return DocID;
  }

  public void setDocID(int docID) {
    DocID = docID;
  }

  public int getDocCount() {
    return DocCount;
  }

  public void setDocCount(int docCount) {
    DocCount = docCount;
  }

  @Override
  public String toString() {
    return "(" +
            "DocID=" + DocID +
            ", DocCount=" + DocCount +
            ')';
  }
}
```