Noah Buchanan

Problem Set 3
Information Retrieval at 5:25 PM

July 30, 2021

Summary

All input words were truncated to a length of 10. The majority of the words matched from my tokenizer were around an average length of 10 so I figured this would be a good point to stem at. As a result all input query words will be truncated to a length of 10 as well so they actually match something in the inverted index. I could not get the pattern matching class in java to work properly but I found an unorthodox but working approach to match words the same as my tokenzier did for input queries, numbers and others are not stored in the inverted index so they are ignored just as my tokenizer would if it were searching for words, I understand this is a naive approach but my time is becoming limited. A custom hashtable class was used for inserting and retrieving from dict.raf call DiskHashTable, its code will be included in the source code portion.

The runtime of BuildInvertedIndex is $\mathcal{O}((F * 2T) + (F * T) + T)$ to be as precise as possible or could just be represented as $\mathcal{O}(F * 2T)$ where F is the total number of input files, and T is the total amount of matched tokens in the files. The runtime of RunQuery() is $\mathcal{O}(Q * P)$ where Q is the number of query input words and P is the postings list for each query word.

The Space complexity for BuildInvertedIndex is $\mathcal{O}(T)$ For the total amount of unique terms, as we store a global hash table before writing it as dict.raf, the first line from 839 consolidated temp files is negligible in this case. The Space complexity for RunQuery() is $\mathcal{O}(Q + P)$ where Q is the query input words and P is the postings files for all documents that contain one of the query words.

Query Results

```
info03@data:/uafs/nbuchanan/IRPS3> ./query.sh Computer
0017-00749358.html.out
0018-00781264.html.out
0017-00733124.html.out
0013-00545731.html.out
0019-00800647.html.out
0010-00446626.html.out
0017-00737767.html.out
0003-00134792.html.out
0009-00413287.html.out
0013-00556551.html.out
info03@data:/uafs/nbuchanan/IRPS3>
```

```
info03@data:/uafs/nbuchanan/IRPS3> ./query.sh Big Data
0019-00804619.html.out
0008-00341657.html.out
0005-00213235.html.out
0007-00316505.html.out
0016-00696167.html.out
0010-00453474.html.out
0011-00470913.html.out
0002-00099197.html.out
0012-00518640.html.out
0013-00578371.html.out
info03@data:/uafs/nbuchanan/IRPS3>
```

```
info03@data:/uafs/nbuchanan/IRPS3> ./query.sh Big Data and Data Analytics
0009-00401602.html.out
0016-00672314.html.out
0011-00470913.html.out
0018-00773645.html.out
0014-00610837.html.out
0010-00451441.html.out
0012-00544179.html.out
0009-00380916.html.out
0013-00556551.html.out
0019-00810205.html.out
info03@data:/uafs/nbuchanan/IRPS3>
```

```
info03@data:/uafs/nbuchanan/IRPS3> ./query.sh Text Mining
0019-00804619.html.out
0005-00213235.html.out
0007-00316505.html.out
0016-00696167.html.out
0012-00518640.html.out
0013-00578371.html.out
0002-00096970.html.out
0014-00610837.html.out
0005-00223861.html.out
0002-00099103.html.out
info03@data:/uafs/nbuchanan/IRPS3>
```

```
info03@data:/uafs/nbuchanan/IRPS3> ./query.sh uafs.edu
0007-00300393.html.out
0001-00066560.html.out
0004-00189552.html.out
0019-00800647.html.out
0000-00026167.html.out
0014-00597905.html.out
0017-00732930.html.out
0010-00439744.html.out
0014-00598552.html.out
0017-00738448.html.out
info03@data:/uafs/nbuchanan/IRPS3>
```

```
info03@data:/uafs/nbuchanan/IRPS3> ./query.sh Data Science, Artificial Intelligence, and 123 Cake Bakeries
0009-00401602.html.out
0016-00672314.html.out
0011-00470913.html.out
0018-00773645.html.out
0014-00610837.html.out
0015-00670568.html.out
0012-00544179.html.out
0007-00303746.html.out
0017-00715947.html.out
0003-00144350.html.out
info03@data:/uafs/nbuchanan/IRPS3>
```

UAInvertedIndex

```
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
```

```
import java.util.Map.Entry;
import java.util.PriorityQueue;
import java.util.TreeMap;

/*******************************
 * Name: Noah Buchanan Username: info03 Problem Set: PS3 Due Date: July 29, 2021
 *******************************/

public class UAInvertedIndex {

 public static final long DICT_REC_LENGTH = 30;
 public static final long POST_REC_LENGTH = 43;
 public static final int branch = 1000;
 public static HashMap<String, long[]> gh = new HashMap<>();

 public static void main(String[] args) throws Exception {

  BuildInvertedIndex(args[0], args[1]);
 }

 public static void BuildInvertedIndex(String input_dir, String output_dir) thro

  // Initial pass through
  File input = new File(input_dir);
  File[] input_files = input.listFiles();
  int docid = 0;
  ArrayList<String> files = new ArrayList<>();
  long size = 0;

  for (File k : input_files) {

   File[] input_files_inner = k.listFiles();

   for (File f : input_files_inner) {

    files.add(f.getName());
    long total = 0;
    BufferedReader br = new BufferedReader(new FileReader(f.getPath()));
    HashMap<String, Integer> ht = new HashMap<>();
    String line = "";
    while ((line = br.readLine()) != null) {

     String[] split = line.split(" ");
     String word = split[split.length - 1];
     if (word.matches("[a-zA-Z]{1,15}")) {
      word = word.toLowerCase();
```

```
      if (word.length() > 10) {
       word = word.substring(0, 10);
      }

      if (ht.get(word) == null) {
       ht.put(word, 1);
      } else {
       ht.put(word, ht.get(word) + 1);
      }
      total++;
     }
    }
    br.close();
    for (String key : ht.keySet()) {
     if (gh.get(key) == null) {
      long[] postdoc = new long[2];
      postdoc[0] = -1;// postings location placeholder
      postdoc[1] = 1;// doc count
      // swapping the order simply for readability from doc count , posting to p
      // , doc count
      gh.put(key, postdoc);
      // this is simply because hashmap.size() returns an integer instead of lon
      // giving the possibility of overflowing
      size++;
     } else {
      long[] postdoc = gh.get(key);
      postdoc[1]++;
      gh.put(key, postdoc);
     }
    }

    TreeMap<String, Integer> sort = new TreeMap<>();
    sort.putAll(ht);
    BufferedWriter bw = new BufferedWriter(new FileWriter(new StringBuilder().ap
      .append(String.format("%06d", docid)).append(".temp").toString()));
    for (String term : sort.keySet()) {
     bw.write(new StringBuilder().append(String.format("%-10s", term))
       .append(String.format(" %06d", docid))
       .append(String.format(" %.15f", (double) ht.get(term) / total)).toString(
     bw.newLine();
    }
    bw.close();

    docid++;
   }
  }
```

```
 mergeSortFiles(docid);

 String line = "";

 BufferedReader br = new BufferedReader(new FileReader("temp/consolidated.temp"
 RandomAccessFile post = new RandomAccessFile(output_dir + "/post.raf", "rw");
 long posting = 0;
 while ((line = br.readLine()) != null) {
  String[] split = line.split(" ");
  if (gh.get(split[0])[0] == -1) {
   long[] postdoc = gh.get(split[0]);
   postdoc[0] = posting;
   gh.put(split[0], postdoc);
  }
  post.seek(posting * POST_REC_LENGTH);
  StringBuilder build = new StringBuilder();
  double rtf = Double.parseDouble(split[split.length - 1]);
  double idf = Math.log(((double) docid / (double) gh.get(split[0])[1]) + 0.000
  // double idf = (double)gh.get(split[0])[1]/(double)docid;
  build.append(String.format("%-22s", files.get(Integer.parseInt(split[split.le
    .append(String.format(" %.15f", rtf * idf)).append("\n");
  post.writeUTF(build.toString());
  posting++;
 }
 post.close();
 br.close();

 DiskHashTable dict = new DiskHashTable(size * 2, DICT_REC_LENGTH, output_dir,

 for (Entry<String, long[]> entry : gh.entrySet()) {
  StringBuilder build = new StringBuilder();
  build.append(String.format("%-10s", entry.getKey())).append(String.format(" %
    .append(String.format(" %06d", entry.getValue()[1]));
  dict.put(build.toString());
 }
}

public static void mergeSortFiles(int size) throws IOException {

 // 1024 is max amount of files to be open on server
 // we will use 1000 and consolodate into approx 840 consolidated files for rea
 // thing
 PriorityQueue<String> heap = new PriorityQueue<String>(new Comparator<String>(
  public int compare(String i, String j) {
   return i.compareTo(j);
```

```
 }
});
int iterations = (int) Math.ceil((float) size / branch);
int count = 0;
int bwcount = 0;
for (int i = 0; i < iterations; i++) {

 BufferedReader[] brs = new BufferedReader[branch];
 int stop = 0;

 for (int j = 0; j < brs.length; j++) {

  StringBuilder build = new StringBuilder();
  // add .append("temp/") for server
  build.append("temp/").append(String.format("%06d", count)).append(".temp");
  if (count < size) {
   brs[j] = new BufferedReader(new FileReader(build.toString()));
   String line = brs[j].readLine();
   if (line == null) {
    stop++;
   } else {
    heap.add(line);
   }

  } else {
   stop++;
  }
  count++;

 }

 // same here add .append("temp/") for server
 BufferedWriter bw = new BufferedWriter(new FileWriter(new StringBuilder().app
   .append(String.format("%03d", bwcount)).append(".temp").toString()));

 while (stop < brs.length) {

  String temp = heap.remove();
  String[] split = temp.split(" ");
  int doc_num = Integer.parseInt(split[split.length - 2]);

  bw.write(temp);
  bw.newLine();
  if ((temp = brs[doc_num - (i * branch)].readLine()) != null) {
   heap.add(temp);
  } else {
```

```
    brs[doc_num - (i * branch)].close();
    stop++;
   }

  }
  bw.close();
  bwcount++;
 }

 // one more time to consolidate into one file consolidated.temp

 heap = new PriorityQueue<String>(new Comparator<String>() {
  public int compare(String i, String j) {
   return i.compareTo(j);
  }
 });
 count = 0;
 bwcount = 0;
 for (int i = 0; i < iterations; i++) {

  BufferedReader[] brs = new BufferedReader[iterations];
  int stop = 0;

  for (int j = 0; j < brs.length; j++) {

   StringBuilder build = new StringBuilder();
   // add .append("temp/") for server
   build.append("temp/").append(String.format("%03d", j)).append(".temp");
   brs[j] = new BufferedReader(new FileReader(build.toString()));
   String line = brs[j].readLine();
   heap.add(line);

  }

  // same here add .append("temp/") for server
  BufferedWriter bw = new BufferedWriter(
    new FileWriter(new StringBuilder().append("temp/").append("consolidated.tem

  while (stop < brs.length) {

   String temp = heap.remove();
   String[] split = temp.split(" ");
   int doc_num = Integer.parseInt(split[split.length - 2]);

   bw.write(temp);
   bw.newLine();
```

```
    if ((temp = brs[doc_num / branch].readLine()) != null) {
     heap.add(temp);
    } else {
     brs[doc_num / branch].close();
     stop++;
    }

   }
   bw.close();
   bwcount++;
  }
 }

}
```

UAQuery

```
import java.io.RandomAccessFile;
import java.util.ArrayList;
import java.util.HashMap;

/*******************************
 * Name: Noah Buchanan Username: info03 Problem Set: PS3 Due Date: July 29, 2021
 *******************************/

public class UAQuery extends UAInvertedIndex {

 public static void main(String[] args) throws Exception {

  // bad way of tokenizing it into just words but the pattern matcher class woul
  // not work
  ArrayList<String> tokenized = new ArrayList<>();
  for (int i = 0; i < args.length; i++) {
   String line = "";
   for (int j = 0; j < args[i].length(); j++) {
    if ((args[i].charAt(j) >= 65 && args[i].charAt(j) <= 90)
      || args[i].charAt(j) >= 97 && args[i].charAt(j) <= 122) {
     line += args[i].charAt(j);
    } else {
     if (!line.equals(""))
      tokenized.add(line);
     line = "";
    }
   }
   if (!line.equals(""))
```

```
    tokenized.add(line);
  }
  String[] query = new String[tokenized.size()];
  for (int i = 0; i < tokenized.size(); i++) {
   query[i] = tokenized.get(i);
  }

  String[] results = RunQuery(query);
  for (String s : results) {
   System.out.println(s);
  }

 }

 public static String[] RunQuery(String[] query) throws Exception {

  for (int i = 0; i < query.length; i++) {
   if (query[i].length() > 10)
    query[i] = query[i].substring(0, 10);

   query[i] = query[i].toLowerCase();
  }

  RandomAccessFile dict = new RandomAccessFile("output/dict.raf", "rw");
  RandomAccessFile post = new RandomAccessFile("output/post.raf", "rw");
  // MAKE SURE THIS IS FALSE V
  DiskHashTable hash = new DiskHashTable(dict.length() / DICT_REC_LENGTH, DICT_RE
  HashMap<String, HashMap<String, Double>> HMDT = new HashMap<>();

  Double[] RTFIDFQuery = new Double[query.length];
  HashMap<String, Integer> count = new HashMap<>();
  for (String term : query) {
   if (count.get(term) == null) {
    count.put(term, 1);
   } else {
    count.put(term, count.get(term) + 1);
   }
  }

  for (int i = 0; i < query.length; i++) {
   RTFIDFQuery[i] = (double) count.get(query[i]) / (double) count.size(); // jus
   String line = hash.get(query[i]);
   String[] split = line.split(" ");
   String term = split[0];

   RTFIDFQuery[i] *= Math.log(839917 / Long.parseLong(split[split.length - 1]) +
```

```
 long posting = Long.parseLong(split[split.length - 2]);
 long doccount = Long.parseLong(split[split.length - 1].substring(0, split[spl
 for (long j = 0; j < doccount; j++) {
  post.seek(posting * POST_REC_LENGTH + j * POST_REC_LENGTH);
  line = post.readUTF();
  split = line.split(" ");
  String doc = split[0];
  double RI = Double.parseDouble(split[split.length - 1]);

  if (HMDT.get(doc) == null)
   HMDT.put(doc, new HashMap<String, Double>());

  if (HMDT.get(doc).get(term) == null)
   HMDT.get(doc).put(term, RI);

 }
}

HashMap<String, Double> cosvals = new HashMap<>();

for (String doc : HMDT.keySet()) {
 // summation of A_i * B_i
 double numerator = 0;
 // sqrt(summation of A_i^2) * sqrt(summation of B_i^2)
 double denominator = 0;
 // summation of A_i^2
 double abssum1 = 0;
 // summation of B_i^2
 double abssum2 = 0;
 int j = 0;
 for (String term : HMDT.get(doc).keySet()) {
  numerator += HMDT.get(doc).get(term) * RTFIDFQuery[j];
  abssum1 += Math.pow(RTFIDFQuery[j], 2);
  abssum2 += Math.pow(HMDT.get(doc).get(term), 2);
  j++;
 }
 // sqrt(summation of A_i^2)
 abssum1 = Math.sqrt(abssum1);
 // sqrt(summation of B_i^2)
 abssum2 = Math.sqrt(abssum2);
 // sqrt(summation of A_i^2) * sqrt(summation of B_i^2)
 denominator = abssum1 * abssum2;
 // cosine similarity of query and document
 cosvals.put(doc, numerator / denominator);
}
```

```
  dict.close();
  post.close();
  return topK(cosvals, 10);
 }

 public static String[] topK(HashMap<String, Double> map, int k) {

  String[] topk = new String[k];
  int count = 0;
  for (String key : map.keySet()) {
   double max = 0;
   String insert = "";
   for (String key2 : map.keySet()) {
    if (map.get(key2) > max) {
     max = map.get(key2);
     insert = key2;
    }
   }
   map.put(insert, (double) -1000);
   topk[count] = insert;
   if (count >= topk.length - 1)
    return topk;
   count++;
  }
  return topk;
 }

}
```

DiskHashTable

```
import java.io.RandomAccessFile;

/*********************************
 * Name: Noah Buchanan Username: info03 Problem Set: PS3 Due Date: July 29, 2021
 *********************************/

public class DiskHashTable {

 public long size;
 public RandomAccessFile dict;
 public long rec_length;

 public DiskHashTable(long size, long rec_length, String output_dir, boolean ini
```

```
 if (init) {
  dict = new RandomAccessFile(output_dir + "/dict.raf", "rw");
  this.rec_length = rec_length;
  this.size = size;

  StringBuilder pad = new StringBuilder();
  pad.append("-1");
  for (int i = 0; i < rec_length - 5; i++) {
   pad.append(" ");
  }
  pad.append("\n");

  for (long i = 0; i < size; i++) {
   dict.seek(i * rec_length);
   dict.writeUTF(pad.toString());
  }
 } else {
  dict = new RandomAccessFile(output_dir + "/dict.raf", "rw");
  this.rec_length = rec_length;
  this.size = size;
 }

}

public void put(String input) throws Exception {

 String[] split = input.split(" ");
 seek(split[0]);
 dict.writeUTF(input);
}

public String get(String input) throws Exception {

 String[] split = input.split(" ");
 seek(split[0]);
 return dict.readUTF();
}

public void seek(String term) throws Exception {

 boolean found = false;
 boolean breaker = false;
 long hash = Math.abs(term.hashCode());
 while (!found && !breaker) {

  long location = ((hash % (size)) * rec_length);
```

```
    dict.seek(location);
    String line = dict.readUTF();

    if (line.split(" ")[0].equals(term)) {
     found = true;
    }

    if (line.split(" ")[0].equals("-1")) {
     breaker = true;
    }

    if (!breaker && !found) {
     hash++;
    } else {
     dict.seek(location);
    }
   }
  }

}
```