CS7070-Big Data Analytics, Spring 2020
Programming Assignment #1
Disha Nagesh Rao, M13254448

Note : For full outputs, please refer the zip folder. For each question, it is under the respective folder >> output.
(tfidf-1/output , tfidf-2/output, tfidf-3/output, tfidf-4/output)

1. (20) Execute all phases of the TFIDF program, on the small sample data shared by Sahil, and submit the following items:
    a. TFIDF for top 18 terms in each document, sorted in descending order of their tfidf values, and formatted for easy readability.

Solution:

PHASE - 1

i. Hadoop com.sun.tools.javac.Main PhaseOne.java
ii. Jar cf PhaseOne.jar PhaseOne*.class
iii. Hadoop jar PhaseOne.jar PhaseOne /tmp/data/input_tfidf.txt /home/maria_dev/tfidf_1_PhaseOne

Source code:

```
1.
    import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseOne
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     private final static IntWritable one = new IntWritable(1);
18.     private Text word = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String doc = value.toString();
23.         String docPart[] = doc.split(" "); //spliting input string to get individual
    words
```

```
24.         String docName = docPart[0]; //getting the document number or the document na
    me
25.         String tempStr=""; //temp string to construct the key part
26.         //loop to collect all the words
27.         //for loop counter i is starting as we have first element of each line as doc
    ument number
28.         for(int i=1;i<docPart.length;i++)
29.         {
30.         tempStr = docPart[i].replaceAll("\\p{P}", ""); //removing special character a
    nd punctuation from the word
31.         tempStr = tempStr+","+docName;
32.         word.set(tempStr);//converting string to text writable
33.         context.write(word,one);
34.         }
35.     }
36.     }
37.
38.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

39.     {
40.
41.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
42.         throws IOException, InterruptedException
43.         {
44.         int sum = 0;
45.         for (IntWritable val : values)
46.         {
47.             sum += val.get();
48.         }
49.         context.write(key, new IntWritable(sum));
50.         }
51.     }
52.
53.     public static void main(String[] args) throws Exception
54.     {
55.     Configuration conf = new Configuration();
56.
57.         Job job = new Job(conf, "PhaseOne");
58.
59.     job.setOutputKeyClass(Text.class);
60.     job.setOutputValueClass(IntWritable.class);
61.     job.setJarByClass(PhaseOne.class);
62.
63.     job.setMapperClass(Map.class);
64.     job.setReducerClass(Reduce.class);
65.
66.     job.setInputFormatClass(TextInputFormat.class);
67.     job.setOutputFormatClass(TextOutputFormat.class);
68.
69.     FileInputFormat.addInputPath(job, new Path(args[0]));
70.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
71.
72.     job.waitForCompletion(true);
73.     }
74.
75. }
```

Output sample :

```
11032,0002  2
11032,0003  1
11032,0009  2
11032,0010  1
11033,0002  1
11033,0003  1
11033,0009  1
11033,0010  1
1500s,0001  1
1500s,0003  1
1500s,0008  1
1500s,0010  1
1914,0003   1
1914,0010   1
1960s,0001  1
1960s,0008  1
200,0005    1
200,0012    1
2000,0002   1
2000,0009   1
45,0002 2
45,0009 2
Aldus,0001  1
Aldus,0008  1
Aliquam,0006    1
Aliquam,0007    1
All,0005    1
All,0012    1
```

PHASE – 2:

i. Hadoop com.sun.tools.javac.Main PhaseTwo.java
ii. Jar cf PhaseTwo.jar PhaseTwo*.class
iii. Hadoop jar PhaseTwo.jar PhaseTwo /home/maria_dev/tfidf_1_PhaseOne /home/maria_dev/tfidf_1_PhaseTwo

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseTwo
```

```
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     //private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String inputLine = value.toString();
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.         String docName = docPart[1]; //getting the document number or the document na
    me
27.         outKey.set(docName);
28.         context.write(outKey,new IntWritable(wordCntr));
29.         //String word = docPart[0];//getting the input word
30.         //String tempStr=""; //temp string to construct the key part
31.
32.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
33.     }
34.     }
35.
36.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

37.     {
38.
39.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
40.         throws IOException, InterruptedException
41.         {
42.         int sum = 0;
43.         for (IntWritable val : values)
44.         {
45.             sum += val.get();
46.         }
47.         context.write(key, new IntWritable(sum));
48.         }
49.     }
50.
51.     public static void main(String[] args) throws Exception
52.     {
53.     Configuration conf = new Configuration();
54.
55.         Job job = new Job(conf, "PhaseTwo");
56.
57.     job.setOutputKeyClass(Text.class);
58.     job.setOutputValueClass(IntWritable.class);
59.     job.setJarByClass(PhaseTwo.class);
60.
61.     job.setMapperClass(Map.class);
62.     job.setReducerClass(Reduce.class);
63.
64.     job.setInputFormatClass(TextInputFormat.class);
65.     job.setOutputFormatClass(TextOutputFormat.class);
66.
67.     FileInputFormat.addInputPath(job, new Path(args[0]));
68.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
69.
```

```
70.     job.waitForCompletion(true);
71.     }
72.
73. }
```

Output:

```
0001    91
0002    129
0003    47
0004    104
0005    121
0006    134
0007    99
0008    91
0009    129
0010    47
0011    104
0012    121
```

PHASE – 3:

i. Hadoop com.sun.tools.javac.Main PhaseThree.java
ii. Jar cf PhaseThree.jar PhaseThree*.class
iii. Hadoop jar PhaseThree.jar PhaseThree /home/maria_dev/tfidf_1_PhaseOne /home/maria_dev/tfidf_1_PhaseThree

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseThree
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
```

```
22.         String inputLine = value.toString(); //input is coming from the output file f
    rom phase one
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.
27.         String word = docPart[0];//getting the input word
28.         outKey.set(word);
29.             context.write(outKey,one);
30.
31.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
32.     }
33.     }
34.
35.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

36.     {
37.
38.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
39.         throws IOException, InterruptedException
40.         {
41.         int sum = 0;
42.         for (IntWritable val : values)
43.         {
44.             sum += val.get();
45.         }
46.         context.write(key, new IntWritable(sum));
47.         }
48.     }
49.
50.     public static void main(String[] args) throws Exception
51.     {
52.     Configuration conf = new Configuration();
53.
54.         Job job = new Job(conf, "PhaseThree");
55.
56.     job.setOutputKeyClass(Text.class);
57.     job.setOutputValueClass(IntWritable.class);
58.     job.setJarByClass(PhaseThree.class);
59.
60.     job.setMapperClass(Map.class);
61.     job.setReducerClass(Reduce.class);
62.
63.     job.setInputFormatClass(TextInputFormat.class);
64.     job.setOutputFormatClass(TextOutputFormat.class);
65.
66.     FileInputFormat.addInputPath(job, new Path(args[0]));
67.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
68.
69.     job.waitForCompletion(true);
70.     }
71.
72. }
```

Output sample:

```
11032   4
11033   4
1500s   4
1914    2
1960s   2
200 2
2000    2
45  2
Aldus   2
Aliquam 2
All 2
BC  2
Bonorum 4
Cicero  4
College 2
Content 2
Contrary    2
Cras    2
Curabitur   1
Curae   1
Donec   2
Duis    1
English 4
Evil    2
Extremes    2
Finibus 4
Good    2
H   2
HampdenSydney   2
```

Paste the PhaseOne, PhaseTwo and PhaseThree outputs in the IDE and create a class 'tfidf' with the attached code (under tfidf.java).
Configure these output files to be the parameters in the following order : PhaseTwo PhaseThree PhaseOne.

Calculating tf_idf and sorting:

I have used the TreeMap data structure, the stream and the collectors library for achieving sorting in the descending order.

```
1.  import sun.reflect.generics.tree.Tree;
2.
3.  import javax.xml.soap.SOAPPart;
4.  import java.util.*;
5.  import java.io.*;
6.  import java.util.concurrent.atomic.AtomicInteger;
7.  import java.util.stream.Collectors;
8.  import java.util.stream.Stream;
9.
10. public class tfidf
11. {
12.
```

```
13. //global variables
14. 
15.     static HashMap<String, Double> tfidf = new HashMap<String, Double>();
16. 
17.     public static void question1a() throws IOException {
18. 
19. 
20.         Map<String,Double> treeMap = new TreeMap<String,Double>(tfidf);
21. //        tfidf.clear();
22.         HashMap<String, Map<String,Double>> test = new HashMap<String, Map<String, Do
    uble>>();
23.         treeMap.entrySet().forEach(entry->{
24.             String key_1 = entry.getKey().split(",")[0];
25.             String key_2 = entry.getKey().split(",")[1];
26.             if(!test.containsKey(key_1)){
27.                 test.put(key_1, new HashMap<String, Double>());
28.             }
29.             test.get(key_1).put(key_2,entry.getValue());
30.         });
31. 
32. 
33.         TreeMap<String, Map<String,Double>> sorted_hashmap = new TreeMap<String, Map<
    String,Double>>(test);
34. 
35.         test.clear();
36. // Question1a
37.         FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/BigD
    ata/HW/PP-1/tfidf-1a/output/tfidf_1a.txt");
38. //Question2a
39. //        FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Bi
    gData/HW/PP-1/tfidf-2/output/tfidf_2.txt");
40. //Question4 - 1a
41. //        FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Bi
    gData/HW/PP-1/tfidf-4/output/question1for4/tfidf_4_1.txt");
42. //Question4 - 2a
43. //        FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Bi
    gData/HW/PP-1/tfidf-4/output/question2for4/tfidf_4_2.txt");
44. 
45.         for (Map.Entry<String, Map<String, Double>> docNumEntry: sorted_hashmap.entry
    Set()) {
46.             String docNum = docNumEntry.getKey();
47. 
48.             Map<String, Double> topTen = null;
49.             for (Map.Entry<String, Double> docTerms : docNumEntry.getValue().entrySet
    ()) {
50.                 String term = docTerms.getKey();
51.                 Double term_tfidf = docTerms.getValue();
52. 
53.                 topTen = docNumEntry.getValue().entrySet().stream()
54.                         .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder())
    )
55.                         .limit(18)
56.                         .collect(Collectors.toMap(
57.                                 Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -
    > e1, LinkedHashMap::new));
58. 
59.             }
60.             sorted_hashmap.replace(docNum, topTen);
61.         }
62.         sorted_hashmap.entrySet().forEach(entry -> {
63.             try{
```

```
64.                 AtomicInteger index = new AtomicInteger();
65.                 writer5.write(entry.getKey() + " = ");
66.                 writer5.write(System.getProperty( "line.separator" ));
67.                 writer5.write("{");
68.                 writer5.write(System.getProperty( "line.separator" ));
69.                 entry.getValue().entrySet().forEach(termEntry->{
70.                     try {
71.                         index.addAndGet(1);
72.                         writer5.write("\t "+ index + ")" +termEntry.getKey() + ": " +
    termEntry.getValue());
73.                         writer5.write(System.getProperty( "line.separator" ));
74.                     } catch (IOException e) {
75.                         e.printStackTrace();
76.                     }
77.                 });
78.                 writer5.write("}");
79.                 writer5.write(System.getProperty( "line.separator" ));
80.                 writer5.write("-----------------------------------------------------
    ---------");
81.                 writer5.write(System.getProperty( "line.separator" ));
82.             } catch (IOException e){
83.                 e.printStackTrace();
84.             }
85.         });
86.         writer5.close();
87.
88.     }
89.
90.     public static <K, V extends Comparable<? super V>> HashMap<K, V> sortByValue(Map<
    K, V> map) {
91.         List<Map.Entry<K, V>> list = new LinkedList<Map.Entry<K, V>>(map.entrySet());

92.         Collections.sort(list,
93.                 new Comparator<Map.Entry<K, V>>() {
94.                     public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {
95.                         return (o1.getValue()).compareTo(o2.getValue());
96.                     }});
97.                     HashMap<K, V> result = new LinkedHashMap<K, V>();
98.                     for (Map.Entry<K, V> entry : list) {
99.                         result.put(entry.getKey(), entry.getValue());
100.                            }
101.                return result;
102.            }
103.
104.            public static void main (String [] args) throws IOException
105.            {
106.                BufferedReader phasetwo = new BufferedReader(new FileReader(args[0]));
    //phasetwo
107.                HashMap<String, Double> idf = new HashMap<String, Double>();
108.                HashMap<String, Integer> tf = new HashMap<String, Integer>();
109.                HashMap<String, Double> tfop = new HashMap<String, Double>();
110.                BufferedReader phasethree = new BufferedReader  (new FileReader(args[1
    ])); //phasethree
111.                BufferedReader phaseone = new BufferedReader  (new FileReader(args[2])
    ); //phaseone
112.
113.                String line;
114.                int count = 0, totalDocs=0, totalTerms=0;
115.
116.       //counting number of documents and simultaneously computing tFrequency value
117.
```

```
118.                while((line=phasetwo.readLine())!= null)
119.                {
120.                    totalDocs++;
121.                    String[] p = line.split("\t", 2);
122.                    String doc = p[0];
123.                    int t = Integer.parseInt(p[1]);
124.                    tf.put(doc, t);
125.                }
126.
127.        //computing idf
128.
129.                while ((line = phasethree.readLine()) != null)
130.                {
131.                    String[] parts = line.split("\t", 2);
132.                    String key = parts[0];
133.                    int value = Integer.parseInt(parts[1]);
134.                    idf.put(key, Math.log10(totalDocs/value));
135.                }
136.
137.        //computing tf
138.
139.                while((line = phaseone.readLine())!= null)
140.                {
141.                    String[] parts = line.split("\t", 2);
142.                    String key = parts[0];
143.                    String docno = key.split(",",2)[1];
144.                    totalTerms = tf.get(docno);
145.                    double value = Integer.parseInt(parts[1]);
146.                    tfop.put(key, value/totalTerms);
147.                }
148.
149.        //computing tfidf
150.
151.                for (String name: tfop.keySet())
152.                {
153.                    String[] key =name.split(",",2);
154.                    String key1 = key[0];
155.                    String key2 = key[1];
156.                    key2 = key2+","+key1;
157.                    double value = tfop.get(name);
158.                    double idff = idf.get(key1);
159.                    tfidf.put(key2, value*idff);
160.
161.                }
162.
163.                tf.clear();
164.                tfop.clear();
165.                idf.clear();
166.
167.                question1a();
168.            }
169.
170.        }
```

Run the program to get the output : tfidf_1a.txt (Please refer attached).

0001 =
{
1)type: 0.017102225283157003
2)dummy: 0.017102225283157003
3)typesetting: 0.017102225283157003
4)with: 0.010486181422410165
5)been: 0.008551112641578502
6)software: 0.008551112641578502
7)release: 0.008551112641578502
8)Aldus: 0.008551112641578502
9)only: 0.008551112641578502
10)1960s: 0.008551112641578502
11)five: 0.008551112641578502
12)centuries: 0.008551112641578502
13)took: 0.008551112641578502
14)including: 0.008551112641578502
15)containing: 0.008551112641578502
16)industrys: 0.008551112641578502
17)printer: 0.008551112641578502
18)survived: 0.008551112641578502
}

0002 =
{
1)45: 0.012064360471064243
2)BC: 0.012064360471064243
3)line: 0.012064360471064243
4)classical: 0.012064360471064243
5)comes: 0.012064360471064243
6)literature: 0.012064360471064243
7)popular: 0.012064360471064243
8)Latin: 0.011095843133015405
9)from: 0.009334263431441277
10)11032: 0.007397228755343603
11)through: 0.006032180235532121
12)very: 0.006032180235532121
13)Contrary: 0.006032180235532121
14)section: 0.006032180235532121
15)during: 0.006032180235532121
16)source: 0.006032180235532121
17)undoubtable: 0.006032180235532121
18)ethics: 0.006032180235532121
}

0003 =
{
1)reproduced: 0.03311281916526143
2)H: 0.016556409582630716
3)chunk: 0.016556409582630716
4)used: 0.016556409582630716
5)Sections: 0.016556409582630716
6)Rackham: 0.016556409582630716
7)1914: 0.016556409582630716
8)those: 0.016556409582630716
9)original: 0.016556409582630716
10)below: 0.016556409582630716
11)accompanied: 0.016556409582630716
12)exact: 0.016556409582630716
13)translation: 0.016556409582630716
14)interested: 0.016556409582630716
15)from: 0.012809787049531115
16)standard: 0.010151516057865158
17)de: 0.010151516057865158
18)Finibus: 0.010151516057865158
}

0004 =
{
1)sometimes: 0.014964447122762379
2)that: 0.014964447122762379
3)readable: 0.014964447122762379
4)using: 0.014964447122762379
5)page: 0.014964447122762379
6)here: 0.014964447122762379
7)content: 0.014964447122762379
8)web: 0.014964447122762379
9)will: 0.014964447122762379
10)like: 0.009175408744608893
11)as: 0.009175408744608893
12)their: 0.009175408744608893
13)reader: 0.0074822235613811895
14)evolved: 0.0074822235613811895
15)sites: 0.0074822235613811895
16)lorem: 0.0074822235613811895
17)Various: 0.0074822235613811895
18)letters: 0.0074822235613811895
}

0005 =
{
1)you: 0.012862004138572622
2)which: 0.012862004138572622
3)Internet: 0.012862004138572622
4)or: 0.012862004138572622
5)words: 0.011829452596355268
6)injected: 0.007886301730903511
7)are: 0.007886301730903511
8)humour: 0.007886301730903511
9)All: 0.006431002069286311
10)slightly: 0.006431002069286311
11)reasonable: 0.006431002069286311
12)available: 0.006431002069286311
13)repetition: 0.006431002069286311
14)predefined: 0.006431002069286311
15)looks: 0.006431002069286311
16)There: 0.006431002069286311
17)etc: 0.006431002069286311
18)generated: 0.006431002069286311
}

0006 =
{
1)semper: 0.02416077416524533
2)nisi: 0.023228395533840106
3)sit: 0.02136363827102966
4)amet: 0.02136363827102966
5)Cras: 0.01742129665038008
6)magna: 0.01742129665038008
7)nunc: 0.016107182776830223
8)Vestibulum: 0.016107182776830223
9)luctus: 0.016107182776830223
10)posuere: 0.016107182776830223
11)Duis: 0.016107182776830223
12)convallis: 0.016107182776830223
13)elit: 0.016107182776830223
14)orci: 0.016107182776830223
15)ultrices: 0.016107182776830223
16)feugiat: 0.011614197766920053
17)ac: 0.011614197766920053
18)eu: 0.011614197766920053
}

0007 =
{
1)augue: 0.023580340920716474
2)diam: 0.023580340920716474
3)ut: 0.021801641334295453
4)fermentum: 0.021801641334295453
5)sollicitudin: 0.021801641334295453
6)imperdiet: 0.01572022728047765
7)mi: 0.01572022728047765
8)Cras: 0.01572022728047765
9)faucibus: 0.01572022728047765
10)viverra: 0.01572022728047765
11)non: 0.01572022728047765
12)sed: 0.01572022728047765
13)vehicula: 0.01572022728047765
14)nisi: 0.01572022728047765
15)nisl: 0.01572022728047765
16)Donec: 0.01572022728047765
17)gravida: 0.01572022728047765
18)elementum: 0.010900820667147726
}

0008 =
{
1)type: 0.017102225283157003
2)dummy: 0.017102225283157003
3)typesetting: 0.017102225283157003
4)with: 0.010486181422410165
5)been: 0.008551112641578502
6)software: 0.008551112641578502
7)release: 0.008551112641578502
8)Aldus: 0.008551112641578502
9)only: 0.008551112641578502
10)1960s: 0.008551112641578502
11)five: 0.008551112641578502
12)centuries: 0.008551112641578502
13)took: 0.008551112641578502
14)including: 0.008551112641578502
15)containing: 0.008551112641578502
16)industrys: 0.008551112641578502
17)printer: 0.008551112641578502
18)survived: 0.008551112641578502
}

0009 =
{
1)45: 0.012064360471064243
2)BC: 0.012064360471064243
3)line: 0.012064360471064243
4)classical: 0.012064360471064243
5)comes: 0.012064360471064243
6)literature: 0.012064360471064243
7)popular: 0.012064360471064243
8)Latin: 0.011095843133015405
9)from: 0.009934263431441277
10)11032: 0.007397228755343603
11)through: 0.006032180235532121
12)very: 0.006032180235532121
13)Contrary: 0.006032180235532121
14)section: 0.006032180235532121
15)during: 0.006032180235532121
16)source: 0.006032180235532121
17)undoubtable: 0.006032180235532121
18)ethics: 0.006032180235532121
}

0010 =
{
1)reproduced: 0.03311281916526143
2)H: 0.016556409582630716
3)chunk: 0.016556409582630716
4)used: 0.016556409582630716
5)Sections: 0.016556409582630716
6)Rackham: 0.016556409582630716
7)1914: 0.016556409582630716
8)those: 0.016556409582630716
9)original: 0.016556409582630716
10)below: 0.016556409582630716
11)accompanied: 0.016556409582630716
12)exact: 0.016556409582630716
13)translation: 0.016556409582630716
14)interested: 0.016556409582630716
15)from: 0.012809787049531115
16)standard: 0.010151516057865158
17)de: 0.010151516057865158
18)Finibus: 0.010151516057865158
}

0011 =
{
1)sometimes: 0.014964447122762379
2)that: 0.014964447122762379
3)readable: 0.014964447122762379
4)using: 0.014964447122762379
5)page: 0.014964447122762379
6)here: 0.014964447122762379
7)content: 0.014964447122762379
8)web: 0.014964447122762379
9)will: 0.014964447122762379
10)like: 0.009175408744608893
11)as: 0.009175408744608893
12)their: 0.009175408744608893
13)reader: 0.0074822235613811895
14)evolved: 0.0074822235613811895
15)sites: 0.0074822235613811895
16)lorem: 0.0074822235613811895
17)Various: 0.0074822235613811895
18)letters: 0.0074822235613811895
}

0012 =
{
1)you: 0.012862004138572622
2)which: 0.012862004138572622
3)Internet: 0.012862004138572622
4)or: 0.012862004138572622
5)words: 0.011829452596355268
6)injected: 0.007886301730903511
7)are: 0.007886301730903511
8)humour: 0.007886301730903511
9)All: 0.006431002069286311
10)slightly: 0.006431002069286311
11)reasonable: 0.006431002069286311
12)available: 0.006431002069286311
13)repetition: 0.006431002069286311
14)predefined: 0.006431002069286311
15)looks: 0.006431002069286311
16)There: 0.006431002069286311
17)etc: 0.006431002069286311
18)generated: 0.006431002069286311
}

---

2. (25) Modify the programs to remove from consideration all those words that occur only once or twice in each document. Repeat the task of Q1 above.

   a. Comment on any changes in the results of part 1(a).

Solution:

The significant changes we can observe are :

- The total number of terms that appear in the result set when terms that appear once/twice are cut off.
- The tf_idf values of common terms in the same document of both the implementations.
- The drastically reduced Phase-3 results.

The terms that appear once/twice, intuitively, are not as significant as the ones which appear in a higher frequency. The 2nd implementation tries to derive those terms that have an imperative effect in the document than the 1st.

PhaseOne_2.java
PhaseTwo_2.java
PhaseThree_2.java

PHASE – 1:

i. Hadoop com.sun.tools.javac.Main PhaseOne_2.java
ii.Jar cf PhaseOne_2.jar PhaseOne_2*.class
iii. Hadoop jar PhaseOne_2.jar PhaseOne_2 /tmp/data/input_tfidf.txt /home/maria_dev/tfidf_1_PhaseOne_2

Source code:

```
1. import java.io.IOException;
2. import java.util.*;
3.
4. import org.apache.hadoop.fs.Path;
5. import org.apache.hadoop.conf.*;
6. import org.apache.hadoop.io.*;
7. import org.apache.hadoop.mapreduce.*;
8. import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

```
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseOne_2
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     private final static IntWritable one = new IntWritable(1);
18.     private Text word = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String doc = value.toString();
23.         String docPart[] = doc.split(" "); //spliting input string to get individual
    words
24.         String docName = docPart[0]; //getting the document number or the document na
    me
25.         String tempStr=""; //temp string to construct the key part
26.         //loop to collect all the words
27.         //for loop counter i is starting as we have first element of each line as doc
    ument number
28.         for(int i=1;i<docPart.length;i++)
29.         {
30.         tempStr = docPart[i].replaceAll("\\p{P}", ""); //removing special character a
    nd punctuation from the word
31.         tempStr = tempStr+","+docName;
32.         word.set(tempStr);//converting string to text writable
33.         context.write(word,one);
34.         }
35.     }
36.     }
37.
38.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

39.     {
40.
41.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
42.         throws IOException, InterruptedException
43.         {
44.         int sum = 0;
45.         for (IntWritable val : values)
46.         {
47.             sum += val.get();
48.         }
49.         if(sum > 2){
50.             context.write(key, new IntWritable(sum));
51.         }
52.
53.         }
54.     }
55.
56.     public static void main(String[] args) throws Exception
57.     {
58.     Configuration conf = new Configuration();
59.
60.         Job job = new Job(conf, "PhaseOne_2");
61.
62.     job.setOutputKeyClass(Text.class);
63.     job.setOutputValueClass(IntWritable.class);
```

```
64.     job.setJarByClass(PhaseOne_2.class);
65.
66.     job.setMapperClass(Map.class);
67.     job.setReducerClass(Reduce.class);
68.
69.     job.setInputFormatClass(TextInputFormat.class);
70.     job.setOutputFormatClass(TextOutputFormat.class);
71.
72.     FileInputFormat.addInputPath(job, new Path(args[0]));
73.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
74.
75.     job.waitForCompletion(true);
76.     }
77.
78. }
```

Output sample:

```
1   Cras,0006   3
2   Ipsum,0001  4
3   Ipsum,0002  4
4   Ipsum,0005  5
5   Ipsum,0008  4
6   Ipsum,0009  4
7   Ipsum,0012  5
8   Latin,0002  3
9   Latin,0009  3
10  Lorem,0001  4
11  Lorem,0002  5
12  Lorem,0005  5
13  Lorem,0008  4
14  Lorem,0009  5
15  Lorem,0012  5
16  a,0002  5
17  a,0004  5
18  a,0005  3
19  a,0009  5
20  a,0011  5
21  a,0012  3
22  amet,0006   6
23  and,0001    3
24  and,0002    3
25  and,0004    3
26  and,0008    3
27  and,0009    3
28  and,0011    3
29  augue,0007  3
30  by,0003 3
31  by,0010 3
32  diam,0007   3
33  et,0006 3
```

PHASE – 2:

i. Hadoop com.sun.tools.javac.Main PhaseTwo_2.java
ii.Jar cf PhaseTwo_2.jar PhaseTwo_2*.class
iii. Hadoop jar PhaseTwo_2.jar PhaseTwo_2
/home/maria_dev/tfidf_1_PhaseOne_2
/home/maria_dev/tfidf_1_PhaseTwo_2

Source code:

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseTwo_2
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     //private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String inputLine = value.toString();
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.         String docName = docPart[1]; //getting the document number or the document na
    me
27.         outKey.set(docName);
28.         context.write(outKey,new IntWritable(wordCntr));
29.         //String word = docPart[0];//getting the input word
30.         //String tempStr=""; //temp string to construct the key part
31.
32.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
33.     }
34.     }
35.
36.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

37.     {
38.
39.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
40.         throws IOException, InterruptedException
41.         {
42.         int sum = 0;
```

```
43.         for (IntWritable val : values)
44.         {
45.             sum += val.get();
46.         }
47.         context.write(key, new IntWritable(sum));
48.         }
49.     }
50.
51.     public static void main(String[] args) throws Exception
52.     {
53.     Configuration conf = new Configuration();
54.
55.         Job job = new Job(conf, "PhaseTwo_2");
56.
57.     job.setOutputKeyClass(Text.class);
58.     job.setOutputValueClass(IntWritable.class);
59.     job.setJarByClass(PhaseTwo_2.class);
60.
61.     job.setMapperClass(Map.class);
62.     job.setReducerClass(Reduce.class);
63.
64.     job.setInputFormatClass(TextInputFormat.class);
65.     job.setOutputFormatClass(TextOutputFormat.class);
66.
67.     FileInputFormat.addInputPath(job, new Path(args[0]));
68.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
69.
70.     job.waitForCompletion(true);
71.     }
72.
73. }
```

Output:

```
1   0001    21
2   0002    42
3   0003    3
4   0004    14
5   0005    32
6   0006    31
7   0007    6
8   0008    21
9   0009    42
10  0010    3
11  0011    14
12  0012    32
13
```

PHASE – 3:

i. Hadoop com.sun.tools.javac.Main PhaseThree_2.java
ii.Jar cf PhaseThree_2.jar PhaseThree_2*.class
iii. Hadoop jar PhaseThree_2.jar PhaseThree_2 /home/maria_dev/tfidf_1_PhaseOne_2 /home/maria_dev/tfidf_1_PhaseThree_2

Source code:

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseThree_2
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String inputLine = value.toString(); //input is coming from the output file f
    rom phase one
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.
27.         String word = docPart[0];//getting the input word
28.         outKey.set(word);
29.             context.write(outKey,one);
30.
31.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
32.     }
33.     }
34.
35.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

36.     {
```

```
37.
38.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
39.         throws IOException, InterruptedException
40.         {
41.         int sum = 0;
42.         for (IntWritable val : values)
43.         {
44.             sum += val.get();
45.         }
46.         context.write(key, new IntWritable(sum));
47.         }
48.     }
49.
50.     public static void main(String[] args) throws Exception
51.     {
52.     Configuration conf = new Configuration();
53.
54.         Job job = new Job(conf, "PhaseThree_2");
55.
56.     job.setOutputKeyClass(Text.class);
57.     job.setOutputValueClass(IntWritable.class);
58.     job.setJarByClass(PhaseThree_2.class);
59.
60.     job.setMapperClass(Map.class);
61.     job.setReducerClass(Reduce.class);
62.
63.     job.setInputFormatClass(TextInputFormat.class);
64.     job.setOutputFormatClass(TextOutputFormat.class);
65.
66.     FileInputFormat.addInputPath(job, new Path(args[0]));
67.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
68.
69.     job.waitForCompletion(true);
70.     }
71.
72. }
```

Output:

```
1   Cras    1
2   Ipsum   6
3   Latin   2
4   Lorem   6
5   a   6
6   amet    1
7   and 6
8   augue   1
9   by  2
10  diam    1
11  et  1
12  from    2
13  in  3
14  magna   1
15  nisi    1
16  of  8
17  semper  1
18  sit 1
19  the 8
20  to  2
21  words   2
```

Calculating tf_idf:

Source code same as above except this change ->

```
1.  // Question1a
2.  //       FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Big
    Data/HW/PP-1/tfidf-1a/output/tfidf_1a.txt");
3.  //Question2a
4.          FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/BigD
    ata/HW/PP-1/tfidf-2/output/tfidf_2.txt");
5.  //Question4 - 1a
6.  //        FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Bi
    gData/HW/PP-1/tfidf-4/output/question1for4/tfidf_4_1.txt");
7.  //Question4 - 2a
8.  //        FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Bi
    gData/HW/PP-1/tfidf-4/output/question2for4/tfidf_4_2.txt");
```

0001 =
{
Lorem: 0.057339046793139274
Ipsum: 0.057339046793139274
and: 0.043004285094854454
the: 0.0
of: 0.0
}

0002 =
{
from: 0.07410964289368034
in: 0.07167380849142409
Latin: 0.05558223217026026
a: 0.035836904245712044
Lorem: 0.035836904245712044
Ipsum: 0.028669523396569637
and: 0.021502142547427227
the: 0.0
of: 0.0
}

0003 =
{
by: 0.7781512503836436
}

```
0004 =
{
   a: 0.10751071273713614
   and: 0.06450642764228168
   the: 0.0
   of: 0.0
}
---------------------------------------------------------------
0005 =
{
   to: 0.09726890629795545
   words: 0.07295167972346658
   Lorem: 0.047035936822497064
   Ipsum: 0.047035936822497064
   a: 0.028221562093498236
   the: 0.0
   of: 0.0
}
---------------------------------------------------------------
0006 =
{
   amet: 0.2088737895576048
   sit: 0.2088737895576048
   nisi: 0.13924919303840322
   Cras: 0.1044368947788024
   magna: 0.1044368947788024
   semper: 0.1044368947788024
   et: 0.1044368947788024
   in: 0.05826387012851249
}
---------------------------------------------------------------
0007 =
{
   augue: 0.5395906230238124
   diam: 0.5395906230238124
}
---------------------------------------------------------------
0008 =
{
   Lorem: 0.057339046793139274
   Ipsum: 0.057339046793139274
   and: 0.043004285094854454
   the: 0.0
   of: 0.0
```

```
}
------------------------------------------------------------
0009 =
{
   from: 0.07410964289368034
   in: 0.07167380849142409
   Latin: 0.05558223217026026
   a: 0.035836904245712044
   Lorem: 0.035836904245712044
   Ipsum: 0.028669523396569637
   and: 0.021502142547427227
   the: 0.0
   of: 0.0
}
------------------------------------------------------------
0010 =
{
   by: 0.7781512503836436
}
------------------------------------------------------------
0011 =
{
   a: 0.10751071273713614
   and: 0.06450642764228168
   the: 0.0
   of: 0.0
}
------------------------------------------------------------
0012 =
{
   to: 0.09726890629795545
   words: 0.07295167972346658
   Lorem: 0.047035936822497064
   Ipsum: 0.047035936822497064
   a: 0.028221562093498236
   the: 0.0
   of: 0.0
}
------------------------------------------------------------

------------------------------------------------------------------------------------------------------------------------
```

b. Select at least 3 different words for which there is a change in their tfidf values and explain the reason for the change.

Solution:

Document 0009, consider the term 'from':
Question1) from: 0.009934263431441277
Question2) from: 0.07410964289368034

Document 0012
Question1) words: 0.011829452596355268
Question2) words: 0.07295167972346658

Document 0006
Question1) magna: 0.01742129665038008
Question2) magna: 0.1044368947788024

tf(t,d) = count of 't' in document 'd' / total number of terms in document 'd'
idf(t)= log10(total number of documents/number of documents with term 't' in it)

tf_idf(t,d) = tf(t,d) * idf(t).

I) The tf(t,d) value for question2 increases because of the lower value of the denominator-> there are a lot less terms in each document than the original total terms in question1.

II) The idf(t) value also increases for question2 because the value of the denominator reduces. (number of documents with term 't' in it will decrease as some documents may have had only 1 or 2 occurrences of term 't' in them, which were removed).

Since both the factors, tf(t,d) and idf(t) increase for Question2, the product, tf_idf(t,d) also increases.

Only terms that have a higher impact overall appear in Question2.

3. (30) Now consider a "Term" to mean a *2*-gram (two words occurring sequentially) in a document. Modify the programs given to you to compute the TFIDF for each 2-gram. Submit the following items:
    a. List of top 20 2-grams for each document, having the highest TFIDF values. The task of selecting the top 20 terms does not need to be done by the MapReduce program.

Solution:

PHASE – 1:

i. Hadoop com.sun.tools.javac.Main PhaseOne_3.java
ii. Jar cf PhaseOne_3.jar PhaseOne_3*.class
iii. Hadoop jar PhaseOne_3.jar PhaseOne_3 /tmp/data/input_tfidf.txt /home/maria_dev/tfidf_1_PhaseOne_3

Source code :

```
import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class PhaseOne_3
{
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException
, InterruptedException
    {
        String doc = value.toString();
        String docPart[] = doc.split(" "); //spliting input string to get individual
words
        String docName = docPart[0]; //getting the document number or the document na
me
```

```
25.         String firstTerm="",SecondTerm=""; //temp string to construct the key part
26.         //loop to collect all the words
27.         //for loop counter i is starting as we have first element of each line as doc
    ument number
28.         for(int i=1;i+1<docPart.length;i++)
29.         {
30.         firstTerm = docPart[i].replaceAll("\\p{P}",""); //removing special character
    and punctuation from the word
31.         SecondTerm = docPart[i+1].replaceAll("\\p{P}","");
32.         firstTerm = firstTerm+"-"+SecondTerm+","+docName;
33.         word.set(firstTerm);//converting string to text writable
34.         context.write(word,one);
35.         }
36.     }
37.     }
38.
39.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

40.     {
41.
42.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
43.         throws IOException, InterruptedException
44.         {
45.         int sum = 0;
46.         for (IntWritable val : values)
47.         {
48.             sum += val.get();
49.         }
50.         context.write(key, new IntWritable(sum));
51.         }
52.     }
53.
54.     public static void main(String[] args) throws Exception
55.     {
56.     Configuration conf = new Configuration();
57.
58.         Job job = new Job(conf, "PhaseOne_3");
59.
60.     job.setOutputKeyClass(Text.class);
61.     job.setOutputValueClass(IntWritable.class);
62.     job.setJarByClass(PhaseOne_3.class);
63.
64.     job.setMapperClass(Map.class);
65.     job.setReducerClass(Reduce.class);
66.
67.     job.setInputFormatClass(TextInputFormat.class);
68.     job.setOutputFormatClass(TextOutputFormat.class);
69.
70.     FileInputFormat.addInputPath(job, new Path(args[0]));
71.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
72.
73.     job.waitForCompletion(true);
74.     }
75.
76. }
```

Output sample:

```
11032-and,0002  1
11032-and,0003  1
11032-and,0009  1
11032-and,0010  1
11033-from,0003 1
11033-from,0010 1
11033-of,0002   1
11033-of,0009   1
1500s-is,0003   1
1500s-is,0010   1
1500s-when,0001 1
1500s-when,0008 1
1914-translation,0003   1
1914-translation,0010   1
1960s-with,0001 1
1960s-with,0008 1
200-Latin,0005  1
200-Latin,0012  1
2000-years,0002 1
2000-years,0009 1
45-BC,0002  2
45-BC,0009  2
Aldus-PageMaker,0001    1
Aldus-PageMaker,0008    1
Aliquam-consectetur,0006    1
Aliquam-varius,0007 1
All-the,0005    1
All-the,0012    1
BC-This,0002    1
BC-This,0009    1
BC-making,0002  1
```

PHASE – 2:

i. Hadoop com.sun.tools.javac.Main PhaseTwo_3.java
ii. Jar cf PhaseTwo_3.jar PhaseTwo_3*.class
iii. Hadoop jar PhaseTwo_3.jar PhaseTwo_3 /home/maria_dev/tfidf_1_PhaseOne_3 /home/maria_dev/tfidf_1_PhaseTwo_3

Source code :

```
1. import java.io.IOException;
2. import java.util.*;
3.
4. import org.apache.hadoop.fs.Path;
5. import org.apache.hadoop.conf.*;
6. import org.apache.hadoop.io.*;
7. import org.apache.hadoop.mapreduce.*;
```

```
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseTwo_3
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     //private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String inputLine = value.toString();
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.         String docName = docPart[1]; //getting the document number or the document na
    me
27.         outKey.set(docName);
28.         context.write(outKey,new IntWritable(wordCntr));
29.         //String word = docPart[0];//getting the input word
30.         //String tempStr=""; //temp string to construct the key part
31.
32.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
33.     }
34.     }
35.
36.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

37.     {
38.
39.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
40.         throws IOException, InterruptedException
41.         {
42.         int sum = 0;
43.         for (IntWritable val : values)
44.         {
45.             sum += val.get();
46.         }
47.         context.write(key, new IntWritable(sum));
48.         }
49.     }
50.
51.     public static void main(String[] args) throws Exception
52.     {
53.     Configuration conf = new Configuration();
54.
55.         Job job = new Job(conf, "PhaseTwo_3");
56.
57.     job.setOutputKeyClass(Text.class);
58.     job.setOutputValueClass(IntWritable.class);
59.     job.setJarByClass(PhaseTwo_3.class);
60.
61.     job.setMapperClass(Map.class);
62.     job.setReducerClass(Reduce.class);
63.
```

```
64.     job.setInputFormatClass(TextInputFormat.class);
65.     job.setOutputFormatClass(TextOutputFormat.class);
66.
67.     FileInputFormat.addInputPath(job, new Path(args[0]));
68.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
69.
70.     job.waitForCompletion(true);
71.     }
72.
73. }
```

Output sample:

1	0001	90
2	0002	128
3	0003	46
4	0004	103
5	0005	120
6	0006	133
7	0007	98
8	0008	90
9	0009	128
10	0010	46
11	0011	103
12	0012	120

PHASE – 3:

i. Hadoop com.sun.tools.javac.Main PhaseThree_3.java
ii. Jar cf PhaseThree_3.jar PhaseThree_3*.class
iii. Hadoop jar PhaseThree_3.jar PhaseThree_3 /home/maria_dev/tfidf_1_PhaseOne_3 /home/maria_dev/tfidf_1_PhaseThree_3

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
12.
13. public class PhaseThree_3
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String inputLine = value.toString(); //input is coming from the output file f
    rom phase one
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.
27.         String word = docPart[0];//getting the input word
28.         outKey.set(word);
29.             context.write(outKey,one);
30.
31.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
32.     }
33.     }
34.
35.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

36.     {
37.
38.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
39.         throws IOException, InterruptedException
40.         {
41.         int sum = 0;
42.         for (IntWritable val : values)
43.         {
44.             sum += val.get();
45.         }
46.         context.write(key, new IntWritable(sum));
47.         }
48.     }
49.
50.     public static void main(String[] args) throws Exception
51.     {
52.     Configuration conf = new Configuration();
53.
54.         Job job = new Job(conf, "PhaseThree_3");
55.
56.     job.setOutputKeyClass(Text.class);
57.     job.setOutputValueClass(IntWritable.class);
58.     job.setJarByClass(PhaseThree_3.class);
59.
60.     job.setMapperClass(Map.class);
61.     job.setReducerClass(Reduce.class);
62.
63.     job.setInputFormatClass(TextInputFormat.class);
64.     job.setOutputFormatClass(TextOutputFormat.class);
65.
66.     FileInputFormat.addInputPath(job, new Path(args[0]));
67.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
68.
69.     job.waitForCompletion(true);
70.     }
71.
72. }
```

Output sample:

```
1   11032-and   4
2   11033-from  2
3   11033-of    2
4   1500s-is    2
5   1500s-when  2
6   1914-translation    2
7   1960s-with  2
8   200-Latin   2
9   2000-years  2
10  45-BC   2
11  Aldus-PageMaker 2
12  Aliquam-consectetur 1
13  Aliquam-varius  1
14  All-the 2
15  BC-This 2
16  BC-making   2
17  Bonorum-et  4
18  Cicero-are  2
19  Cicero-written  2
20  College-in  2
21  Content-here    2
22  Contrary-to 2
23  Cras-aliquet    1
24  Cras-et 2
25  Cras-eu 1
26  Cras-imperdiet  1
27  Curabitur-elementum 1
28  Curae-Suspendisse   1
29  Donec-euismod   1
30  Donec-feugiat   1
31  Donec-imperdiet 1
```

Calculating tf_idf and sorting:

```
1.  import java.util.*;
2.  import java.io.*;
3.  import java.util.concurrent.atomic.AtomicInteger;
4.  import java.util.stream.Collectors;
5.
6.  public class tfidf_3
7.  {
8.
```

```
9.  //global variables
10. 
11.     static HashMap<String, Double> tfidf = new HashMap<String, Double>();
12.     static ArrayList<String> sortedKeys;
13.     static HashMap<String, Double> tfidf1b = new HashMap<String, Double>();
14. 
15. //function to segregate the tf idf values based on document number with words in asce
    nding order
16. 
17.     public static void question3a() throws IOException {
18.         // Disha change - begin
19. 
20.         Map<String,Double> treeMap = new TreeMap<String,Double>(tfidf);
21.         HashMap<String, Map<String,Double>> test = new HashMap<String, Map<String, Do
    uble>>();
22.         treeMap.entrySet().forEach(entry->{
23.             String key_1 = entry.getKey().split(",")[0];
24.             String key_2 = entry.getKey().split(",")[1];
25.             if(!test.containsKey(key_1)){
26.                 test.put(key_1, new HashMap<String, Double>());
27.             }
28.             test.get(key_1).put(key_2,entry.getValue());
29.         });
30. 
31. 
32.         TreeMap<String, Map<String,Double>> sorted_hashmap = new TreeMap<String, Map<
    String,Double>>(test);
33. 
34.         test.clear();
35.         treeMap.clear();
36. // Question 4 - 3a
37.         FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/BigD
    ata/HW/PP-1/tfidf-4/output/question3for4/tfidf_4_3.txt");
38. // Question 3a
39. //          FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/B
    igData/HW/PP-1/tfidf-3a/tfidf_3.txt");
40. 
41.         for (Map.Entry<String, Map<String, Double>> docNumEntry: sorted_hashmap.entry
    Set()) {
42.             String docNum = docNumEntry.getKey();
43. 
44.             Map<String, Double> topTen = null;
45.             for (Map.Entry<String, Double> docTerms : docNumEntry.getValue().entrySet
    ()) {
46.                 String term = docTerms.getKey();
47.                 Double term_tfidf = docTerms.getValue();
48. 
49.                 topTen = docNumEntry.getValue().entrySet().stream()
50.                         .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder())
    )
51.                         .limit(20)
52.                         .collect(Collectors.toMap(
53.                                 Map.Entry::getKey, Map.Entry::getValue, (e1, e2) -
    > e1, LinkedHashMap::new));
54. 
55.             }
56.             sorted_hashmap.replace(docNum, topTen);
57.         }
58. 
59.         final Boolean[] proceed = {true};
60.         System.out.println("Sorted_hashmap created!");
```

```
61.         sorted_hashmap.entrySet().forEach(entry -> {
62.             try{
63.                 if(proceed[0]){
64.                     AtomicInteger index = new AtomicInteger();
65.                     writer5.write(entry.getKey() + " = ");
66.                     writer5.write(System.getProperty( "line.separator" ));
67.                     writer5.write("{");
68.                     writer5.write(System.getProperty( "line.separator" ));
69.                     entry.getValue().entrySet().forEach(termEntry->{
70.
71.                             try {
72.                                 index.addAndGet(1);
73.                                 writer5.write("\t "+ index + ")" + termEntry.getKey()
     + ": " + termEntry.getValue());
74.                                 writer5.write(System.getProperty( "line.separator" ))
    ;
75.                             } catch (IOException e) {
76.                                 e.printStackTrace();
77.                             }
78.
79.                     });
80.                     writer5.write("}");
81.                     writer5.write(System.getProperty( "line.separator" ));
82.                     writer5.write("-----------------------------------------------
    -------------");
83.                     writer5.write(System.getProperty( "line.separator" ));
84.                 }
85.
86.             } catch (IOException e){
87.                 e.printStackTrace();
88.             }
89.         });
90.         writer5.close();
91.
92.         // Disha change - end
93.
94. //         sortedKeys = new ArrayList<String>(tfidf.keySet());
95. //         try{
96. //             FileWriter writer = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM
    /BigData/HW/output/sorted.txt");
97. //
98. ////actual sorting step
99. //
100.        //             Collections.sort(sortedKeys);
101.        //             for (String x : sortedKeys)
102.        //             {
103.        //                 writer.write(x + " " + tfidf.get(x) + "\n");
104.        //                 tfidf1b.put(x, tfidf.get(x));
105.        //             }
106.        //             writer.close();
107.        //         }
108.        //         catch ( IOException e )
109.        //         {
110.        //             e.printStackTrace();
111.        //         }
112.            }
113.
114.            public static <K, V extends Comparable<? super V>> HashMap<K, V> sortByVal
    ue(Map<K, V> map) {
115.                List<Map.Entry<K, V>> list = new LinkedList<Map.Entry<K, V>>(map.entry
    Set());
```

```
116.                 Collections.sort(list,
117.                         new Comparator<Map.Entry<K, V>>() {
118.                             public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2)
    {
119.                                 return (o1.getValue()).compareTo(o2.getValue());
120.                             }});
121.                 HashMap<K, V> result = new LinkedHashMap<K, V>();
122.                 for (Map.Entry<K, V> entry : list) {
123.                     result.put(entry.getKey(), entry.getValue());
124.                 }
125.                 return result;
126.             }
127.
128.             public static void main (String [] args) throws IOException
129.             {
130.                 BufferedReader phasetwo = new BufferedReader(new FileReader(args[0]));
    //phasetwo
131.                 HashMap<String, Double> idf = new HashMap<String, Double>();
132.                 HashMap<String, Integer> tf = new HashMap<String, Integer>();
133.                 HashMap<String, Double> tfop = new HashMap<String, Double>();
134.                 BufferedReader phasethree = new BufferedReader  (new FileReader(args[1
    ])); //phasethree
135.                 BufferedReader phaseone = new BufferedReader  (new FileReader(args[2])
    ); //phaseone
136.
137.                 String line;
138.                 int count = 0, totalDocs=0, totalTerms=0;
139.
140.         //counting number of documents and simultaneously computing tFrequency value
141.
142.                 while((line=phasetwo.readLine())!= null)
143.                 {
144.                     totalDocs++;
145.                     String[] p = line.split("\t", 2);
146.                     String doc = p[0];
147.                     int t = Integer.parseInt(p[1]);
148.                     tf.put(doc, t);
149.                 }
150.
151.         //computing idf
152.
153.                 while ((line = phasethree.readLine()) != null)
154.                 {
155.                     String[] parts = line.split("\t", 2);
156.                     String key = parts[0];
157.                     int value = Integer.parseInt(parts[1]);
158.                     idf.put(key, Math.log10(totalDocs/value));
159.                 }
160.
161.         //computing tf
162.
163.                 while((line = phaseone.readLine())!= null)
164.                 {
165.                     String[] parts = line.split("\t", 2);
166.                     String key = parts[0];
167.                     String docno = key.split(",",2)[1];
168.                     totalTerms = tf.get(docno);
169.                     double value = Integer.parseInt(parts[1]);
170.                     tfop.put(key, value/totalTerms);
171.                 }
172.
```

```
173.        //computing tfidf
174.
175.                for (String name: tfop.keySet())
176.                {
177.                    String[] key =name.split(",",2);
178.                    String key1 = key[0];
179.                    String key2 = key[1];
180.                    key2 = key2+","+key1;
181.                    double value = tfop.get(name);
182.                    double idff = idf.get(key1);
183.                    tfidf.put(key2, value*idff);
184.
185.        //sorting in the required order
186.
187.                }
188.                question3a();
189.
190.        //        tf.clear();
191.        //        idf.clear();
192.        //        tfop.clear();
193.            }
194.
195.        }
```

Output :

0001 =
{
1)dummy-text: 0.017292250008525415
2)into-electronic: 0.008646125004262708
3)only-five: 0.008646125004262708
4)typesetting-industry: 0.008646125004262708
5)remaining-essentially: 0.008646125004262708
6)essentially-unchanged: 0.008646125004262708
7)and-scrambled: 0.008646125004262708
8)also-the: 0.008646125004262708
9)software-like: 0.008646125004262708
10)text-of: 0.008646125004262708
11)a-type: 0.008646125004262708
12)the-printing: 0.008646125004262708
13)but-also: 0.008646125004262708
14)leap-into: 0.008646125004262708
15)and-more: 0.008646125004262708
16)printing-and: 0.008646125004262708
17)containing-Lorem: 0.008646125004262708
18)industry-Lorem: 0.008646125004262708
19)the-1960s: 0.008646125004262708
20)popularised-in: 0.008646125004262708
}

0002 =
{
1)comes-from: 0.012158613287244432
2)from-a: 0.012158613287244432
3)45-BC: 0.012158613287244432
4)of-the: 0.0074550196049947256
5)very-popular: 0.006079306643622216
6)College-in: 0.006079306643622216
7)Richard-McClintock: 0.006079306643622216
8)more-obscure: 0.006079306643622216
9)Extremes-of: 0.006079306643622216
10)has-roots: 0.006079306643622216
11)line-in: 0.006079306643622216
12)Malorum-The: 0.006079306643622216
13)from-45: 0.006079306643622216
14)a-Latin: 0.006079306643622216
15)ethics-very: 0.006079306643622216
16)discovered-the: 0.006079306643622216
17)HampdenSydney-College: 0.006079306643622216
18)sections-11032: 0.006079306643622216
19)McClintock-a: 0.006079306643622216
20)is-not: 0.006079306643622216
}

0003 =
{
1)form-accompanied: 0.01691633153007921
2)Cicero-are: 0.01691633153007921
3)reproduced-in: 0.01691633153007921
4)for-those: 0.01691633153007921
5)versions-from: 0.01691633153007921
6)H-Rackham: 0.01691633153007921
7)11033-from: 0.01691633153007921
8)The-standard: 0.01691633153007921
9)Sections-11032: 0.01691633153007921
10)original-form: 0.01691633153007921
11)standard-chunk: 0.01691633153007921
12)used-since: 0.01691633153007921
13)from-de: 0.01691633153007921
14)also-reproduced: 0.01691633153007921
15)below-for: 0.01691633153007921
16)translation-by: 0.01691633153007921
17)the-1914: 0.01691633153007921

18)is-reproduced: 0.01691633153007921
19)their-exact: 0.01691633153007921
20)interested-Sections: 0.01691633153007921
}

0004 =
{
1)by-the: 0.007554866508579064
2)web-page: 0.007554866508579064
3)humour-and: 0.007554866508579064
4)using-Content: 0.007554866508579064
5)has-a: 0.007554866508579064
6)content-here: 0.007554866508579064
7)it-look: 0.007554866508579064
8)a-moreorless: 0.007554866508579064
9)normal-distribution: 0.007554866508579064
10)ipsum-will: 0.007554866508579064
11)when-looking: 0.007554866508579064
12)publishing-packages: 0.007554866508579064
13)use-Lorem: 0.007554866508579064
14)long-established: 0.007554866508579064
15)distribution-of: 0.007554866508579064
16)fact-that: 0.007554866508579064
17)Ipsum-as: 0.007554866508579064
18)text-and: 0.007554866508579064
19)moreorless-normal: 0.007554866508579064
20)is-that: 0.007554866508579064
}

0005 =
{
1)the-Internet: 0.01296918750639406
2)humour-or: 0.01296918750639406
3)on-the: 0.007952020911994373
4)injected-humour: 0.007952020911994373
5)randomised-words: 0.00648459375319703
6)sentence-structures: 0.00648459375319703
7)dictionary-of: 0.00648459375319703
8)free-from: 0.00648459375319703
9)majority-have: 0.00648459375319703
10)sure-there: 0.00648459375319703
11)to-use: 0.00648459375319703
12)true-generator: 0.00648459375319703
13)you-need: 0.00648459375319703

14)always-free: 0.00648459375319703
15)with-a: 0.00648459375319703
16)even-slightly: 0.00648459375319703
17)need-to: 0.00648459375319703
18)look-even: 0.00648459375319703
19)generator-on: 0.00648459375319703
20)first-true: 0.00648459375319703
}

0006 =
{
1)sit-amet: 0.021524267130210335
2)orci-pharetra: 0.008114144707125
3)ornare-commodo: 0.008114144707125
4)nunc-in: 0.008114144707125
5)Suspendisse-vulputate: 0.008114144707125
6)posuere-cubilia: 0.008114144707125
7)cubilia-Curae: 0.008114144707125
8)in-consequat: 0.008114144707125
9)semper-mollis: 0.008114144707125
10)Pellentesque-malesuada: 0.008114144707125
11)Duis-magna: 0.008114144707125
12)eu-mi: 0.008114144707125
13)nunc-quam: 0.008114144707125
14)amet-consectetur: 0.008114144707125
15)justo-magna: 0.008114144707125
16)mollis-arcu: 0.008114144707125
17)Vestibulum-ante: 0.008114144707125
18)Aliquam-consectetur: 0.008114144707125
19)Nulla-feugiat: 0.008114144707125
20)quam-semper: 0.008114144707125
}

0007 =
{
1)porta-faucibus: 0.011012053531098213
2)nisi-ut: 0.011012053531098213
3)placerat-tortor: 0.011012053531098213
4)mi-est: 0.011012053531098213
5)gravida-pulvinar: 0.011012053531098213
6)magna-id: 0.011012053531098213
7)eros-gravida: 0.011012053531098213
8)aliquet-facilisis: 0.011012053531098213
9)nulla-Cras: 0.011012053531098213

10)tincidunt-fermentum: 0.011012053531098213
11)Sed-non: 0.011012053531098213
12)pulvinar-nisi: 0.011012053531098213
13)vitae-leo: 0.011012053531098213
14)sodales-fermentum: 0.011012053531098213
15)ut-lacinia: 0.011012053531098213
16)volutpat-ante: 0.011012053531098213
17)nisl-nibh: 0.011012053531098213
18)facilisi-Sed: 0.011012053531098213
19)sed-Nunc: 0.011012053531098213
20)Aliquam-varius: 0.011012053531098213
}

0008 =
{
1)dummy-text: 0.017292250008525415
2)into-electronic: 0.008646125004262708
3)only-five: 0.008646125004262708
4)typesetting-industry: 0.008646125004262708
5)remaining-essentially: 0.008646125004262708
6)essentially-unchanged: 0.008646125004262708
7)and-scrambled: 0.008646125004262708
8)also-the: 0.008646125004262708
9)software-like: 0.008646125004262708
10)text-of: 0.008646125004262708
11)a-type: 0.008646125004262708
12)the-printing: 0.008646125004262708
13)but-also: 0.008646125004262708
14)leap-into: 0.008646125004262708
15)and-more: 0.008646125004262708
16)printing-and: 0.008646125004262708
17)containing-Lorem: 0.008646125004262708
18)industry-Lorem: 0.008646125004262708
19)the-1960s: 0.008646125004262708
20)popularised-in: 0.008646125004262708
}

0009 =
{
1)comes-from: 0.012158613287244432
2)from-a: 0.012158613287244432
3)45-BC: 0.012158613287244432
4)of-the: 0.0074550196049947256
5)very-popular: 0.006079306643622216

6)College-in: 0.006079306643622216
7)Richard-McClintock: 0.006079306643622216
8)more-obscure: 0.006079306643622216
9)Extremes-of: 0.006079306643622216
10)has-roots: 0.006079306643622216
11)line-in: 0.006079306643622216
12)Malorum-The: 0.006079306643622216
13)from-45: 0.006079306643622216
14)a-Latin: 0.006079306643622216
15)ethics-very: 0.006079306643622216
16)discovered-the: 0.006079306643622216
17)HampdenSydney-College: 0.006079306643622216
18)sections-11032: 0.006079306643622216
19)McClintock-a: 0.006079306643622216
20)is-not: 0.006079306643622216
}

0010 =
{
1)form-accompanied: 0.01691633153007921
2)Cicero-are: 0.01691633153007921
3)reproduced-in: 0.01691633153007921
4)for-those: 0.01691633153007921
5)versions-from: 0.01691633153007921
6)H-Rackham: 0.01691633153007921
7)11033-from: 0.01691633153007921
8)The-standard: 0.01691633153007921
9)Sections-11032: 0.01691633153007921
10)original-form: 0.01691633153007921
11)standard-chunk: 0.01691633153007921
12)used-since: 0.01691633153007921
13)from-de: 0.01691633153007921
14)also-reproduced: 0.01691633153007921
15)below-for: 0.01691633153007921
16)translation-by: 0.01691633153007921
17)the-1914: 0.01691633153007921
18)is-reproduced: 0.01691633153007921
19)their-exact: 0.01691633153007921
20)interested-Sections: 0.01691633153007921
}

0011 =
{
1)by-the: 0.007554866508579064

2)web-page: 0.007554866508579064
3)humour-and: 0.007554866508579064
4)using-Content: 0.007554866508579064
5)has-a: 0.007554866508579064
6)content-here: 0.007554866508579064
7)it-look: 0.007554866508579064
8)a-moreorless: 0.007554866508579064
9)normal-distribution: 0.007554866508579064
10)ipsum-will: 0.007554866508579064
11)when-looking: 0.007554866508579064
12)publishing-packages: 0.007554866508579064
13)use-Lorem: 0.007554866508579064
14)long-established: 0.007554866508579064
15)distribution-of: 0.007554866508579064
16)fact-that: 0.007554866508579064
17)Ipsum-as: 0.007554866508579064
18)text-and: 0.007554866508579064
19)moreorless-normal: 0.007554866508579064
20)is-that: 0.007554866508579064
}

0012 =
{
1)the-Internet: 0.01296918750639406
2)humour-or: 0.01296918750639406
3)on-the: 0.007952020911994373
4)injected-humour: 0.007952020911994373
5)randomised-words: 0.00648459375319703
6)sentence-structures: 0.00648459375319703
7)dictionary-of: 0.00648459375319703
8)free-from: 0.00648459375319703
9)majority-have: 0.00648459375319703
10)sure-there: 0.00648459375319703
11)to-use: 0.00648459375319703
12)true-generator: 0.00648459375319703
13)you-need: 0.00648459375319703
14)always-free: 0.00648459375319703
15)with-a: 0.00648459375319703
16)even-slightly: 0.00648459375319703
17)need-to: 0.00648459375319703
18)look-even: 0.00648459375319703
19)generator-on: 0.00648459375319703
20)first-true: 0.00648459375319703
}

---

b. Which output – obtained in 3(a) or in 2(a) – better characterizes the documents? Give reasons for your answers.

Solution: The output obtained in 3(a) better characterizes the document as it captures the essential details of the document with context. For instance, '*normal-distribution'* as 2 separate distinct terms have a tf-idf value associated with them. But, when estimated as bigrams, 'normal-distribution' have a weightage and context associated with it.
In addition, removal of terms with frequency less than 2 will not result in inconsistent interpretation of data in bigrams as in unigrams. This yields better results.

---

4. (20) Once your program is working for the above two parts, run the programs on a larger collection of documents (to be provided to you by March 2nd) and repeat the above task . Discuss the results for 1(a), 2(a), and 3(a) in the context of the new set of documents.

Solution:

Question #	Hadoop Commands
Question 4 for 1:	i. Hadoop com.sun.tools.javac.Main PhaseOne_4.java ii. Jar cf PhaseOne_4.jar PhaseOne_4*.class iii. Hadoop jar PhaseOne_4.jar PhaseOne_4 /tmp/data/input_tfidf.txt /home/maria_dev/tfidf_1_PhaseOne_4 --- i. Hadoop com.sun.tools.javac.Main PhaseTwo_4.java ii. Jar cf PhaseTwo_4.jar PhaseTwo_4*.class iii. Hadoop jar PhaseTwo_4.jar PhaseTwo_4 /home/maria_dev/tfidf_1_PhaseOne_4 /home/maria_dev/tfidf_1_PhaseTwo_4 ---

	i. Hadoop com.sun.tools.javac.Main PhaseThree_4.java ii. Jar cf PhaseThree_4.jar PhaseThree_4*.class iii. Hadoop jar PhaseThree_4.jar PhaseThree_4 /home/maria_dev/tfidf_1_PhaseOne_4 /home/maria_dev/tfidf_1_PhaseThree_4
Question 4 for 2:	i. Hadoop com.sun.tools.javac.Main PhaseOne_4_2q.java ii. Jar cf PhaseOne_4_2q.jar PhaseOne_4_2q*.class iii. Hadoop jar PhaseOne_4_2q.jar PhaseOne_4_2q /tmp/data/input_tfidf.txt /home/maria_dev/tfidf_1_PhaseOne_4_2q ---------- i. Hadoop com.sun.tools.javac.Main PhaseTwo_4_2q.java ii. Jar cf PhaseTwo_4_2q.jar PhaseTwo_4_2q*.class iii. Hadoop jar PhaseTwo_4_2q.jar PhaseTwo_4_2q /home/maria_dev/tfidf_1_PhaseOne_4_2q /home/maria_dev/tfidf_1_PhaseTwo_4_2q i. Hadoop com.sun.tools.javac.Main PhaseThree_4_2q.java ii. Jar cf PhaseThree_4_2q.jar PhaseThree_4_2q*.class iii. Hadoop jar PhaseThree_4_2q.jar PhaseThree_4_2q /home/maria_dev/tfidf_1_PhaseOne_4_2q /home/maria_dev/tfidf_1_PhaseThree_4_2q
Question 4 for 3:	i. Hadoop com.sun.tools.javac.Main PhaseOne_4_3q.java ii. Jar cf PhaseOne_4_3q.jar PhaseOne_4_3q*.class

	iii. Hadoop jar PhaseOne_4_3q.jar PhaseOne_4_3q /tmp/data/input_tfidf.txt /home/maria_dev/tfidf_1_PhaseOne_4_3q --- i. Hadoop com.sun.tools.javac.Main PhaseTwo_4_3q.java ii. Jar cf PhaseTwo_4_3q.jar PhaseTwo_4_3q*.class iii. Hadoop jar PhaseTwo_4_3q.jar PhaseTwo_4_3q /home/maria_dev/tfidf_1_PhaseOne_4_3q /home/maria_dev/tfidf_1_PhaseTwo_4_3q --- i. Hadoop com.sun.tools.javac.Main PhaseThree_4_3q.java ii. Jar cf PhaseThree_4_3q.jar PhaseThree_4_3q*.class iii. Hadoop jar PhaseThree_4_3q.jar PhaseThree_4_3q /home/maria_dev/tfidf_1_PhaseOne_4_3q /home/maria_dev/tfidf_1_PhaseThree_4_3q

Outputs :

Calculating tf_idf and sorting: same except for the filewriter block. Choose different files for different data inputs/questions.

I have used the FileSplit library with addInputPaths API to process multiple input paths at once.

```
1.  // Question1a
2.  //       FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Big
    Data/HW/PP-1/tfidf-1a/output/tfidf_1a.txt");
3.  //Question2a
4.  //       FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/Big
    Data/HW/PP-1/tfidf-2/output/tfidf_2.txt");
5.  //Question4 - 1a
6.          FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/BigD
    ata/HW/PP-1/tfidf-4/output/question1for4/tfidf_4_1.txt");
7.  //Question4 - 2a
8.          FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/BigD
    ata/HW/PP-1/tfidf-4/output/question2for4/tfidf_4_2.txt");
9.  // Question 4 - 3a
```

```
10.         FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/BigData/HW/PP-1/tfidf-4/output/question3for4/tfidf_4_3.txt");
11. // Question 3a
12. //         FileWriter writer5 = new FileWriter("C:/Users/Disha/Documents/Disha/3SEM/BigData/HW/PP-1/tfidf-3a/tfidf_3.txt");
13.
```

Question1 for 4:
Phase -1 :

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12. import org.apache.hadoop.mapreduce.lib.input.FileSplit;
13.
14. public class PhaseOne_4
15. {
16.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
17.     {
18.     private final static IntWritable one = new IntWritable(1);
19.     private Text word = new Text();
20.
21.     public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
22.     {
23.
24.
25.         String doc = value.toString();
26.         String docPart[] = doc.split(" "); //splitting input string to get individual words
27.
28.         String docName = ((FileSplit) context.getInputSplit()).getPath().getName().toString();
29.
30.         String tempStr=""; //temp string to construct the key part
31.         //loop to collect all the words
32.         //for loop counter i is starting as we have first element of each line as document number
33.         for(int i=1;i<docPart.length;i++)
34.         {
35.         tempStr = docPart[i].replaceAll("\\p{P}", ""); //removing special character and punctuation from the word
36.         if(tempStr != null && !tempStr.isEmpty()){
```

```
37.                 tempStr = tempStr+","+docName;
38.                 word.set(tempStr);//converting string to text writable
39.                 context.write(word,one);
40.         }
41.
42.         }
43.     }
44.     }
45.
46.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

47.     {
48.
49.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
50.         throws IOException, InterruptedException
51.         {
52.         int sum = 0;
53.         for (IntWritable val : values)
54.         {
55.             sum += val.get();
56.         }
57.         context.write(key, new IntWritable(sum));
58.         }
59.     }
60.
61.     public static void main(String[] args) throws Exception
62.     {
63.     Configuration conf = new Configuration();
64.
65.         Job job = new Job(conf, "PhaseOne_4");
66.
67.     job.setOutputKeyClass(Text.class);
68.     job.setOutputValueClass(IntWritable.class);
69.     job.setJarByClass(PhaseOne_4.class);
70.
71.     job.setMapperClass(Map.class);
72.     job.setReducerClass(Reduce.class);
73.
74.     job.setInputFormatClass(TextInputFormat.class);
75.     job.setOutputFormatClass(TextOutputFormat.class);
76.
77.     FileInputFormat.addInputPaths(job, "/home/maria_dev/question4/74-
    0.txt,/home/maria_dev/question4/76-0.txt,/home/maria_dev/question4/84-0.txt");
78.     FileOutputFormat.setOutputPath(job, new Path(args[0]));
79.
80.     job.waitForCompletion(true);
81.     }
82.
83. }
```

Output sample:

```
$1,74-0.txt 1
$1,76-0.txt 1
$200,76-0.txt   1
$5000,84-0.txt  1
1,74-0.txt  1
1,76-0.txt  1
1,84-0.txt  5
10,76-0.txt 1
10,84-0.txt 2
11,84-0.txt 2
11th,84-0.txt   2
12,84-0.txt 2
12th,84-0.txt   2
13,84-0.txt 3
13th,84-0.txt   1
14,84-0.txt 2
15,84-0.txt 2
1500,74-0.txt   1
1500,76-0.txt   1
1500,84-0.txt   1
16,84-0.txt 2
17,84-0.txt 13
18,84-0.txt 2
1876,74-0.txt   1
18th,84-0.txt   2
19,84-0.txt 2
19th,84-0.txt   1
```

Phase- 2:

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseTwo_4
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     //private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
```

```
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String inputLine = value.toString();
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.         String docName = docPart[1]; //getting the document number or the document na
    me
27.         outKey.set(docName);
28.         context.write(outKey,new IntWritable(wordCntr));
29.         //String word = docPart[0];//getting the input word
30.         //String tempStr=""; //temp string to construct the key part
31.
32.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
33.     }
34.     }
35.
36.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

37.     {
38.
39.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
40.         throws IOException, InterruptedException
41.         {
42.         int sum = 0;
43.         for (IntWritable val : values)
44.         {
45.             sum += val.get();
46.         }
47.         context.write(key, new IntWritable(sum));
48.         }
49.     }
50.
51.     public static void main(String[] args) throws Exception
52.     {
53.     Configuration conf = new Configuration();
54.
55.         Job job = new Job(conf, "PhaseTwo_4");
56.
57.     job.setOutputKeyClass(Text.class);
58.     job.setOutputValueClass(IntWritable.class);
59.     job.setJarByClass(PhaseTwo_4.class);
60.
61.     job.setMapperClass(Map.class);
62.     job.setReducerClass(Reduce.class);
63.
64.     job.setInputFormatClass(TextInputFormat.class);
65.     job.setOutputFormatClass(TextOutputFormat.class);
66.
67.     FileInputFormat.addInputPath(job, new Path(args[0]));
68.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
69.
70.     job.waitForCompletion(true);
71.     }
72.
73. }
```

Output:

```
1   74-0.txt    66925
2   76-0.txt    104697
3   84-0.txt    71404
```

Phase-3 :

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12.
13. public class PhaseThree_4
14. {
15.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
16.     {
17.     private final static IntWritable one = new IntWritable(1);
18.     private Text outKey = new Text();
19.
20.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
21.     {
22.         String inputLine = value.toString(); //input is coming from the output file f
    rom phase one
23.         String temp[] = inputLine.split("\t"); //spliting input string to get pair of
     word,document name and frequency
24.         //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
25.         String docPart[]=temp[0].split(",");//seperating document name and word
26.
27.         String word = docPart[0];//getting the input word
28.         outKey.set(word);
29.             context.write(outKey,one);
30.
31.         //loop is not required in this mapper as we know that the input string will o
    nly have 3 parts
32.     }
33.     }
34.
35.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

36.     {
37.
```

```
38.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
39.         throws IOException, InterruptedException
40.         {
41.         int sum = 0;
42.         for (IntWritable val : values)
43.         {
44.             sum += val.get();
45.         }
46.         context.write(key, new IntWritable(sum));
47.         }
48.     }
49.
50.     public static void main(String[] args) throws Exception
51.     {
52.     Configuration conf = new Configuration();
53.
54.         Job job = new Job(conf, "PhaseThree_4");
55.
56.     job.setOutputKeyClass(Text.class);
57.     job.setOutputValueClass(IntWritable.class);
58.     job.setJarByClass(PhaseThree_4.class);
59.
60.     job.setMapperClass(Map.class);
61.     job.setReducerClass(Reduce.class);
62.
63.     job.setInputFormatClass(TextInputFormat.class);
64.     job.setOutputFormatClass(TextOutputFormat.class);
65.
66.     FileInputFormat.addInputPath(job, new Path(args[0]));
67.     FileOutputFormat.setOutputPath(job, new Path(args[1]));
68.
69.     job.waitForCompletion(true);
70.     }
71.
72. }
```

Output sample:

```
$1  2
$200    1
$5000   1
1   3
10  2
11  1
11th    1
12  1
12th    1
13  1
13th    1
14  1
15  1
1500    3
16  1
17  1
18  1
1876    1
18th    1
19  1
19th    1
1C  3
1E  3
1E1 3
1E7 3
1E8 3
1E9 3
1F3 3
```

74-0.txt =
{
Becky: 5.489478761810086E-4
Potter: 2.566509550976144E-4
Joes: 1.7822982992889895E-4
Muff: 1.354546707459632E-4
hanted: 9.267951156302744E-5
Spaniard: 8.555031836587149E-5
Beckys: 8.555031836587149E-5
auntie: 8.555031836587149E-5
alley: 7.842112516871552E-5
Amy: 7.129193197155957E-5
tickets: 7.129193197155957E-5
lads: 7.129193197155957E-5
Pollys: 7.129193197155957E-5
Welshman: 7.129193197155957E-5
trifle: 7.129193197155957E-5
Cardiff: 6.41627387744036E-5
glanced: 6.41627387744036E-5
Hood: 6.41627387744036E-5

}

76-0.txt =
{
en: 9.387754995104965E-4
duke: 5.787596526108401E-4
dat: 2.9165840761491153E-4
canoe: 2.6887259451999657E-4
dey: 2.3697245618711565E-4
Jane: 1.914008299972857E-4
gwyne: 1.7317217952135373E-4
Buck: 1.5038636642643877E-4
wuz: 1.4582920380745576E-4
ll: 1.4127204118847278E-4
Jims: 1.367148785694898E-4
doan: 1.230433907125408E-4
runaway: 1.1848622809355783E-4
n: 1.1392906547457483E-4
m: 1.0937190285559184E-4
uz: 1.0937190285559184E-4
mo: 1.0481474023660884E-4
Silas: 1.0025757761762586E-4
}

84-0.txt =
{
Elizabeth: 4.8778572061138536E-4
Clerval: 2.8064383925586556E-4
Felix: 2.8064383925586556E-4
cottage: 2.6727984691034815E-4
Justine: 2.605978507375894E-4
Geneva: 2.071418813555198E-4
companion: 1.937778890100024E-4
sensations: 1.870958928372437E-4
beheld: 1.8041389666448497E-4
appearance: 1.7373190049172626E-4
Victor: 1.7373190049172626E-4
fiend: 1.6704990431896756E-4
endeavoured: 1.6036790814620886E-4
science: 1.5368591197345016E-4
greater: 1.5368591197345016E-4
desired: 1.4700391580069148E-4
Safie: 1.4700391580069148E-4
Frankenstein: 1.4700391580069148E-4

}

Question 2 for 4 :

Phase – 1:

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12. import org.apache.hadoop.mapreduce.lib.input.FileSplit;
13.
14. public class PhaseOne_4_2q
15. {
16.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
17.     {
18.     private final static IntWritable one = new IntWritable(1);
19.     private Text word = new Text();
20.
21.     public void map(LongWritable key, Text value, Context context) throws IOException
    , InterruptedException
22.     {
23.         String doc = value.toString();
24.         String docPart[] = doc.split(" "); //splitting input string to get individual
     words
25.
26.         String docName = ((FileSplit) context.getInputSplit()).getPath().getName().to
    String();
27.
28.         String tempStr=""; //temp string to construct the key part
29.         //loop to collect all the words
30.         //for loop counter i is starting as we have first element of each line as doc
    ument number
31.         for(int i=1;i<docPart.length;i++)
32.         {
33.             tempStr = docPart[i].replaceAll("\\p{P}", ""); //removing special charact
    er and punctuation from the word
34.             if(tempStr != null && !tempStr.isEmpty()){
35.                     tempStr = tempStr+","+docName;
36.                     word.set(tempStr);//converting string to text writable
37.                     context.write(word,one);
38.             }
39.
```

```
40.         }
41.     }
42.     }
43.
44.     public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

45.     {
46.
47.     public void reduce(Text key, Iterable<IntWritable> values, Context context)
48.         throws IOException, InterruptedException
49.         {
50.         int sum = 0;
51.         for (IntWritable val : values)
52.         {
53.             sum += val.get();
54.         }
55.         if(sum > 2){
56.             context.write(key, new IntWritable(sum));
57.         }
58.
59.         }
60.     }
61.
62.     public static void main(String[] args) throws Exception
63.     {
64.     Configuration conf = new Configuration();
65.
66.         Job job = new Job(conf, "PhaseOne_4_2q");
67.
68.     job.setOutputKeyClass(Text.class);
69.     job.setOutputValueClass(IntWritable.class);
70.     job.setJarByClass(PhaseOne_4_2q.class);
71.
72.     job.setMapperClass(Map.class);
73.     job.setReducerClass(Reduce.class);
74.
75.     job.setInputFormatClass(TextInputFormat.class);
76.     job.setOutputFormatClass(TextOutputFormat.class);
77.
78.     FileInputFormat.addInputPaths(job, "/home/maria_dev/question4/74-
    0.txt,/home/maria_dev/question4/76-0.txt,/home/maria_dev/question4/84-0.txt");
79.     FileOutputFormat.setOutputPath(job, new Path(args[0]));
80.
81.     job.waitForCompletion(true);
82.     }
83.
84. }
```

Output sample:

```
1,84-0.txt  5
13,84-0.txt 3
17,84-0.txt 13
1E1,74-0.txt    4
1E1,76-0.txt    4
1E1,84-0.txt    4
1E8,74-0.txt    3
1E8,76-0.txt    3
1E8,84-0.txt    3
1F3,74-0.txt    3
1F3,76-0.txt    3
1F3,84-0.txt    3
2,74-0.txt  10
2,84-0.txt  5
20,84-0.txt 3
3,74-0.txt  3
3,76-0.txt  4
3,84-0.txt  7
4,74-0.txt  3
4,76-0.txt  4
4,84-0.txt  7
5,84-0.txt  3
7th,84-0.txt    3
A,74-0.txt  47
A,76-0.txt  39
A,84-0.txt  32
ANY,84-0.txt    3
Abner,76-0.txt  5
About,74-0.txt  3
About,76-0.txt  8
About,84-0.txt  3
Adventures,74-0.txt 4
Adventures,76-0.txt 3
```

Phase – 2:

Source code : Same as Question2 , Phase - 2

Output sample:

```
74-0.txt    59445
76-0.txt    98156
84-0.txt    65586
```

Phase – 3:

Source code : Same as Question2, Phase -3

Output sample:

```
1   1    1
2   13   1
3   17   1
4   1E1  3
5   1E8  3
6   1F3  3
7   2    2
8   20   1
9   3    3
10  4    3
11  5    1
12  7th  1
13  A    3
14  ANY  1
15  Abner    1
16  About    3
17  Adventures   2
18  After    3
19  Again    1
20  Agatha   1
21  Agrippa  1
22  Ah   2
23  Aint     1
24  Alas     2
25  Alfred   1
26  All  3
27  Alps     1
28  Amen     1
29  America  1
```

74-0.txt =
{
Becky: 6.180223166526034E-4
Potter: 2.889454986947237E-4
Joes: 2.0065659631578032E-4
Muff: 1.5249901319999305E-4
Hucks: 1.5249901319999305E-4
treasure: 1.4447274934736184E-4
box: 1.2039395778946819E-4
hanted: 1.0434143008420577E-4
finally: 9.631516623157456E-5
Spaniard: 9.631516623157456E-5
Beckys: 9.631516623157456E-5
tavern: 9.631516623157456E-5
minister: 9.631516623157456E-5

Saturday: 9.631516623157456E-5
auntie: 9.631516623157456E-5
bury: 8.828890237894334E-5
later: 8.828890237894334E-5
alley: 8.828890237894334E-5
}

76-0.txt =
{
en: 0.0010013343908905259
king: 6.562142852923349E-4
duke: 6.17327512830567E-4
dat: 3.11094179694144E-4
canoe: 2.8678994690553897E-4
o: 2.6734656067465494E-4
dey: 2.52764021001492E-4
niggers: 2.28459788212887E-4
Sally: 2.1873809509744498E-4
er: 2.04155555424282E-4
Jane: 2.04155555424282E-4
gwyne: 1.8471216919339797E-4
Buck: 1.6040793640479298E-4
amongst: 1.6040793640479298E-4
wuz: 1.55547089847072E-4
ll: 1.5068624328935099E-4
Jims: 1.4582539673162998E-4
gun: 1.40964550173909E-4
}

84-0.txt =
{
whom: 5.601551644163999E-4
Elizabeth: 5.310561948363272E-4
Clerval: 3.055391805907636E-4
Felix: 3.055391805907636E-4
cottage: 2.9098969580072727E-4
despair: 2.8371495340570905E-4
Justine: 2.8371495340570905E-4
ice: 2.691654686156727E-4
affection: 2.618907262206545E-4
mountains: 2.4006649903559997E-4
Geneva: 2.255170142455636E-4
journey: 2.1824227185054544E-4
companion: 2.1096752945552727E-4

possessed: 2.1096752945552727E-4
existence: 2.0369278706050908E-4
sensations: 2.0369278706050908E-4
monster: 2.0369278706050908E-4
countenance: 1.9641804466549089E-4
}

Question 3 for 4:

Phase – 1:

Source code :

```
1.  import java.io.IOException;
2.  import java.util.*;
3.
4.  import org.apache.hadoop.fs.Path;
5.  import org.apache.hadoop.conf.*;
6.  import org.apache.hadoop.io.*;
7.  import org.apache.hadoop.mapreduce.*;
8.  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9.  import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
10. import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
12. import org.apache.hadoop.mapreduce.lib.input.FileSplit;
13.
14. public class PhaseOne_4_3q
15. {
16.     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
17.     {
18.         private final static IntWritable one = new IntWritable(1);
19.         private Text word = new Text();
20.
21.         public void map(LongWritable key, Text value, Context context) throws IOExcep
    tion, InterruptedException
22.         {
23.             String doc = value.toString();
24.             String docPart[] = doc.split(" "); //splitting input string to get indivi
    dual words
25.
26.             String docName = ((FileSplit) context.getInputSplit()).getPath().getName(
    ).toString();
27.
28.             String firstTerm="",SecondTerm=""; //temp string to construct the key par
    t
29.             //loop to collect all the words
30.             //for loop counter i is starting as we have first element of each line as
     document number
31.             for(int i=1;i+1<docPart.length;i++)
32.             {
33.                 firstTerm = docPart[i].replaceAll("\\p{P}",""); //removing special ch
    aracter and punctuation from the word
```

```
                SecondTerm = docPart[i+1].replaceAll("\\p{P}","");
                if(firstTerm != null && !firstTerm.isEmpty()){
                    if(SecondTerm != null && !SecondTerm.isEmpty()){
                        firstTerm = firstTerm+"-"+SecondTerm+","+docName;
                        word.set(firstTerm);//converting string to text writable
                        context.write(word,one);
                    }
                }
            }
        }
    }

    public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable>

    {

    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException
        {
        int sum = 0;
        for (IntWritable val : values)
        {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
        }
    }

    public static void main(String[] args) throws Exception
    {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "PhaseOne_4_3q");

    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    job.setJarByClass(PhaseOne_4_3q.class);

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPaths(job, "/home/maria_dev/question4/74-
0.txt,/home/maria_dev/question4/76-0.txt,/home/maria_dev/question4/84-0.txt");
    FileOutputFormat.setOutputPath(job, new Path(args[0]));

    job.waitForCompletion(true);
    }

}
```

Output sample:

```
$1-to,74-0.txt  1
$1-to,76-0.txt  1
$200-reward,76-0.txt
$5000-are,84-0.txt  1
1-General,74-0.txt  1
1-General,76-0.txt  1
10-cents,76-0.txt   1
11th-17,84-0.txt    1
11th-the,84-0.txt   1
12th-17,84-0.txt    1
13-2018,84-0.txt    1
13th-17,84-0.txt    1
1500-West,74-0.txt  1
1500-West,76-0.txt  1
1500-West,84-0.txt  1
17-2008,84-0.txt    1
18th-17,84-0.txt    2
19th-17,84-0.txt    1
1C-below,84-0.txt   1
1E-below,74-0.txt   1
1E-below,76-0.txt   1
1E-below,84-0.txt   1
1E1-through,74-0.txt
1E1-through,76-0.txt
1E1-through,84-0.txt
1E1-with,74-0.txt   1
1E1-with,76-0.txt   1
1E1-with,84-0.txt   1
1E7-and,74-0.txt    1
1E7-and,76-0.txt    1
1E7-and,84-0.txt    1
1E7-or,84-0.txt 1
1E8-or,74-0.txt 2
```

Phase – 2:

Source code : Same as Question3, Phase -2

Output sample:

```
74-0.txt    60231
76-0.txt    92134
84-0.txt    63264
```

Phase – 3:

Source code : Same as Question3, Phase -3

Output sample:

```
$1-to   2
$200-reward 1
$5000-are   1
1-General   2
10-cents    1
11th-17 1
11th-the    1
12th-17 1
13-2018 1
13th-17 1
1500-West   3
17-2008 1
18th-17 1
19th-17 1
1C-below    1
1E-below    3
1El-through 3
1El-with    3
1E7-and 3
1E7-or  1
1E8-or  3
1F3-a   3
1F3-the 3
1F3-this    3
2-Information   2
2-and   1
2-is    1
2-nipped    1
2-that  1
2-the   1
```

74-0.txt =
{

1)Injun-Joe: 3.01017875833826E-4
2)said-Huck: 1.1882284572387866E-4
3)Injun-Joes: 1.109013226756201E-4
4)it-He: 1.0297979962736152E-4
5)said-Joe: 1.0297979962736152E-4
6)Muff-Potter: 1.0297979962736152E-4
7)and-Huck: 8.713675353084436E-5
8)Huck-was: 8.713675353084436E-5
9)Joe-Harper: 8.713675353084436E-5
10)boys-were: 7.921523048258578E-5
11)the-knife: 7.921523048258578E-5
12)on-Tom: 7.12937074343272E-5
13)No-2: 7.12937074343272E-5
14)Aunt-Pollys: 7.12937074343272E-5
15)the-treasure: 7.12937074343272E-5

16)and-Becky: 7.12937074343272E-5
17)in-order: 7.12937074343272E-5
18)they-came: 7.12937074343272E-5
19)Becky-Thatcher: 6.337218438606863E-5
20)aint-any: 6.337218438606863E-5
}

76-0.txt =
{
1)the-king: 5.074986754349851E-4
2)he-says: 4.2464174883335493E-4
3)the-duke: 3.935704013577435E-4
4)warnt-no: 3.1589203266871524E-4
5)begun-to: 3.1589203266871524E-4
6)the-canoe: 2.2267799024188123E-4
7)says-the: 1.86428084853668E-4
8)gwyne-to: 1.86428084853668E-4
9)Mary-Jane: 1.812495269410661E-4
10)Aunt-Sally: 1.6571385320326046E-4
11)we-went: 1.5535673737805667E-4
12)and-didnt: 1.449996215528529E-4
13)was-pretty: 1.39821063640251E-4
14)Jim-was: 1.1910683198984344E-4
15)she-says: 1.1910683198984344E-4
16)in-de: 1.1910683198984344E-4
17)canoe-and: 1.1910683198984344E-4
18)could-a: 1.0874971616463966E-4
19)Jim-said: 1.0357115825203778E-4
20)he-come: 1.0357115825203778E-4
}

84-0.txt =
{
1)whom-I: 3.2429523825470226E-4
2)I-shall: 2.639612404398739E-4
3)the-cottage: 2.2625249180560625E-4
4)upon-me: 1.7346024371763145E-4
5)of-man: 1.5837674426392436E-4
6)me-from: 1.5837674426392436E-4
7)I-cannot: 1.5083499453707083E-4
8)endeavoured-to: 1.5083499453707083E-4
9)unable-to: 1.3575149508336375E-4
10)my-dear: 1.2820974535651021E-4
11)my-friends: 1.2820974535651021E-4

12)the-fiend: 1.2066799562965667E-4
13)I-resolved: 1.2066799562965667E-4
14)the-cause: 1.1312624590280313E-4
15)of-ice: 1.1312624590280313E-4
16)at-length: 1.0558449617594958E-4
17)I-became: 1.0558449617594958E-4
18)death-of: 1.0558449617594958E-4
19)yet-I: 1.0558449617594958E-4
20)my-journey: 9.804274644909604E-5
}

Term	Frequency – 1 and 2	Frequency > 2
Becky | 74-0.txt	5.489478761810086E-4	6.180223166526034E-4
En | 76-0.txt	9.387754995104965E-4	0. 0010013343908905259
Clerval | 84-0.txt	2.8064383925586556E-4	3.055391805907636E-4

As discussed earlier, the values are increased for frequency > 2 (Question 2) because of the number of terms in the document and the count of its occurrences in each document.

However, those terms which are of high tf-idf values appear in the result set of top 18 terms. These terms are those that hold high importance in the context of those specific documents.

The ones that are a part of the result set from Question1, but not a part of the result set from Question 2 are those terms that haven't appeared in that particular document more than twice, but, might have appeared in others making its idf value higher and thus appearing in the result set.

Bigram vs unigram:

The tf_idf values are significantly lower (in scale of e-5). This happens because each term (except the first and the last word of the document) occurrence is a subset of 2 different word-pairs (bigrams).
For example, '*The adventure of*' word will result in "The-adventure" and "adventure-of", where the term 'adventure' appears twice. The total number of word-pairs is almost double the sum of terms in each document and hence, with the denominator (total number of terms) increasing, the overall value drops drastically.

But, bigrams have context and convey the meaning of the usage much better than unigrams, as unigrams have no context associated with them. So, although a few unigrams may have a staggering tf_idf value, its context is hidden.
The tf_idf values of bigrams, on the other hand, have much more meaning as they express the

context better. Hence, a high tf_idf value of bigram portrays the importance of the word pair better.