Code Used:

Phase One for Program 3:

```java
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
{
    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 name
        String tempStr=""; //temp string to construct the key part
        String term ="";
        String term1="";
        //loop to collect all the words
        //for loop counter i is starting as we have first element of each line as document number
        String prev=" ";
        for(int i=1;i<docPart.length;i++)
        {
        prev = docPart[i-1].replaceAll("\\p{P}", ""); //removing special character and punctuation from the word
        tempStr = docPart[i].replaceAll("\\p{P}", ""); //removing special character and punctuation from the word
        term = prev+" "+tempStr;
        term1 = term+","+docName;
        //word.set(term1.toUpperCase());//converting string to text writable
         word.set(term1);
               context.write(word,one);
        }
    }
    }

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

```

```java
    public void reduce(Text key, Iterable<IntWritable> values, Context context)
        throws IOException, InterruptedException
        {
        int sum = 0;
        for (IntWritable val : values)
        {
            sum += val.get();
        }
             // if(sum>1)
               {
        context.write(key, new IntWritable(sum));
        }
           }
    }

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

        //Job job = new Job(conf, "PhaseOne");

        Job job = Job.getInstance(conf, "PhaseOne");


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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Two for Program 3:

```java
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 PhaseTwo
{
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
    //private final static IntWritable one = new IntWritable(1);
    private Text outKey = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String inputLine = value.toString();
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word,document name and frequency
        int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word
        String docName = docPart[1]; //getting the document number or the document name

        outKey.set(docName);
        context.write(outKey,new IntWritable(wordCntr));
        //String word = docPart[0];//getting the input word
        //String tempStr=""; //temp string to construct the key part

        //loop is not required in this mapper as we know that the input string will only have 3 parts
    }
    }

    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, "PhaseTwo");
        Job job = Job.getInstance(conf, "PhaseTwo");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Three for Program 3:

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

    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
    {
        String inputLine = value.toString(); //input is coming from the output file from phase one
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word,document name and frequency
        //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word

        String word = docPart[0].toUpperCase();//getting the input word
        outKey.set(word);
            context.write(outKey,one);

        //loop is not required in this mapper as we know that the input string will only have 3 parts
    }
    }

    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));
```

```java
        }
    }

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

        //Job job = new Job(conf, "PhaseThree");
        Job job = Job.getInstance(conf, "PhaseThree");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase One for Program 4 Part A (Without filter):

```java
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;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;


public class PhaseOne
{
    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
    {
  // Getting the file name as the key value
```

```
            String filePathString = ((FileSplit) context.getInputSplit()).getPath().get
Name().toString();

            filePathString = filePathString.substring(0, filePathString.indexOf("."));


            String line = value.toString().replaceAll("\\p{P}", "").toUpperCase();
            StringTokenizer tokenizer = new StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {

          String filepathword = tokenizer.nextToken()+ "," +filePathString ;
          word.set(filepathword);
          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");

        Job job = Job.getInstance(conf, "PhaseOne");


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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Two for Program 4(All parts):

```
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 PhaseTwo
{
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
    //private final static IntWritable one = new IntWritable(1);
    private Text outKey = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, Int
erruptedException
    {
        String inputLine = value.toString();
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word
,document name and frequency
        int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word
        String docName = docPart[1]; //getting the document number or the document name
        outKey.set(docName);
        context.write(outKey,new IntWritable(wordCntr));
        //String word = docPart[0];//getting the input word
        //String tempStr=""; //temp string to construct the key part

        //loop is not required in this mapper as we know that the input string will only h
ave 3 parts
    }
    }

    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, "PhaseTwo");
        Job job = Job.getInstance(conf, "PhaseTwo");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Three for Program 4 (All parts):

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

    public void map(LongWritable key, Text value, Context context) throws IOException, Int
erruptedException
    {
        String inputLine = value.toString(); //input is coming from the output file from p
hase one
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word
,document name and frequency
        //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word

        String word = docPart[0].toUpperCase();//getting the input word
        outKey.set(word);
            context.write(outKey,one);

        //loop is not required in this mapper as we know that the input string will only h
ave 3 parts
    }
    }

    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, "PhaseThree");
        Job job = Job.getInstance(conf, "PhaseThree");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase One for Program 4 Part B (With filter):

```
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;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;


public class PhaseOne
```

```
{
    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, Int
erruptedException
    {
  // Getting the filename as the key value
            String filePathString = ((FileSplit) context.getInputSplit()).getPath().getNam
e().toString();

            filePathString = filePathString.substring(0, filePathString.indexOf("."));

            String line = value.toString().replaceAll("\\p{P}", "").toUpperCase();
            StringTokenizer tokenizer = new StringTokenizer(line);
          while (tokenizer.hasMoreTokens()) {

          String filepathword =  tokenizer.nextToken()+ "," + filePathString;
          word.set(filepathword);
          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();
        }
            if (sum>1)
              {
        context.write(key, new IntWritable(sum));
        }
    }
}

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

        //Job job = new Job(conf, "PhaseOne");

        Job job = Job.getInstance(conf, "PhaseOne");


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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Two for Program 4(Same for All parts):

```
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 PhaseTwo
{
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
    //private final static IntWritable one = new IntWritable(1);
    private Text outKey = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, Int
erruptedException
    {
        String inputLine = value.toString();
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word
,document name and frequency
        int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word
        String docName = docPart[1]; //getting the document number or the document name
        outKey.set(docName);
        context.write(outKey,new IntWritable(wordCntr));
        //String word = docPart[0];//getting the input word
        //String tempStr=""; //temp string to construct the key part

        //loop is not required in this mapper as we know that the input string will only h
ave 3 parts
    }
    }

    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, "PhaseTwo");
        Job job = Job.getInstance(conf, "PhaseTwo");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Three for Program 4 (Same for All parts):

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

    public void map(LongWritable key, Text value, Context context) throws IOException, Int
erruptedException
    {
        String inputLine = value.toString(); //input is coming from the output file from p
hase one
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word
,document name and frequency
```

```
        //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word

        String word = docPart[0].toUpperCase();//getting the input word
        outKey.set(word);
            context.write(outKey,one);

        //loop is not required in this mapper as we know that the input string will only h
ave 3 parts
    }
    }

    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, "PhaseThree");
        Job job = Job.getInstance(conf, "PhaseThree");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase One for Program 4 Part C (Bigrams):

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

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
```

```java
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;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;


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

    public void map(LongWritable key, Text value, Context context) throws IOException, Int
erruptedException
    {
        String filePathString = ((FileSplit) context.getInputSplit()).getPath().getName().
toString();

            filePathString = filePathString.substring(0, filePathString.indexOf("."));

        String prev = null;

            String line = value.toString().replaceAll("\\p{P}", "").toUpperCase();

            StringTokenizer itr = new StringTokenizer(line);

           while (itr.hasMoreTokens())
           {
             String cur = itr.nextToken();

             // Emit only if we have an actual bigram.

             if (prev != null)
             {
               bigram.set(prev + " " + cur+ "," +filePathString);
               context.write(bigram, one);
             }
             prev = cur;
           }
         // String prev1 = prev;
        }

        }


    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();
        }
             // if(sum>1)
               {
```

```
        context.write(key, new IntWritable(sum));
        }
           }
    }

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

        //Job job = new Job(conf, "PhaseOne");

        Job job = Job.getInstance(conf, "PhaseOne");


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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Two for Program 4(Same for All parts):

```
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 PhaseTwo
{
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
    //private final static IntWritable one = new IntWritable(1);
    private Text outKey = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, Int
erruptedException
    {
        String inputLine = value.toString();
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word
,document name and frequency
```

```
        int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word
        String docName = docPart[1]; //getting the document number or the document name
        outKey.set(docName);
        context.write(outKey,new IntWritable(wordCntr));
        //String word = docPart[0];//getting the input word
        //String tempStr=""; //temp string to construct the key part

        //loop is not required in this mapper as we know that the input string will only h
ave 3 parts
    }
    }

    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, "PhaseTwo");
        Job job = Job.getInstance(conf, "PhaseTwo");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Phase Three for Program 4 (Same for All parts):

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

import org.apache.hadoop.fs.Path;
```

```java
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 PhaseThree
{
    public static class Map extends Mapper<LongWritable, Text, Text, IntWritable>
    {
    private final static IntWritable one = new IntWritable(1);
    private Text outKey = new Text();

    public void map(LongWritable key, Text value, Context context) throws IOException, Int
erruptedException
    {
        String inputLine = value.toString(); //input is coming from the output file from p
hase one
        String temp[] = inputLine.split("\t"); //spliting input string to get pair of word
,document name and frequency
        //int wordCntr = Integer.parseInt(temp[1]);//getting word frequency
        String docPart[]=temp[0].split(",");//seperating document name and word

        String word = docPart[0].toUpperCase();//getting the input word
        outKey.set(word);
            context.write(outKey,one);

        //loop is not required in this mapper as we know that the input string will only h
ave 3 parts
    }
    }

    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, "PhaseThree");
        Job job = Job.getInstance(conf, "PhaseThree");

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

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

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

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
    }

}
```

Python Code Used to calculate TF IDF (All programs):

```
# In[2]: IMPORTING THE REQUIRED PACKAGES

import os
import pandas as pd
from io import StringIO
import numpy as np
import math

# In[3]: READING IN THE OUTPUT OF THREE MAP REDUCE PHASES

p1= pd.read_table('C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\output.txt',heade
r=None,names=['term_doc','term_freq'])
p3= pd.read_table('C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\output3.txt',head
er=None,names=['word','occ'])
p2= pd.read_table('C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\output2.txt',head
er=None,names =['doc','freq'],
                 converters={'doc': lambda x: str(x)})

# In[5]:CALCULATING IDF OF A TERM

idf_array =[]
for i, row in enumerate(p3.values):
    idf = math.log(4/row[1])
    idf_array.append(idf)
#print(idf_array)
idf_df=p3.assign(idf=idf_array)
#print(idf_df)

# In[6]:GETTING THE TOTAL NUMBER OF TERMS IN A DOCUMENT VALUE FROM PHASE2 OUTPUT

d1=dict(zip(p2['doc'],p2['freq']))
p1_1=p1.join(p1['term_doc'].str.split(',',1,expand=True).rename(columns={0:'word', 1:'doc'
}))
p1_1['freq']=p1_1.doc.map(d1)


# In[7]: CALCULATING TF OF A TERM
tf_array=[]
for i,row in enumerate(p1_1.values):
    tf=p1_1.term_freq/p1_1.freq
    tf_array.append(tf)
tf_df=p1_1.assign(tf=tf_array[0])

# In[8]:GETTING THE IDF CALCULATED ABOVE
```

```
d2=dict(zip(idf_df['word'],idf_df['idf']))
tf_df['idf']=tf_df.word.map(d2)

# In[9]: CALCULATING THE FINAL TF_IDF OF WORD IN EACH DOCUMENT

tf_idf_array =[]
for i, row in enumerate(tf_df.values):
    tf_idf = tf_df.tf*tf_df.idf
    tf_idf_array.append(tf_idf)

tf_idf_df=tf_df.assign(tf_idf=tf_idf_array[0])


# In[11]: GETTING THE TOP 15 WORDS FOR EACH OF THE DOCUMENTS

final=tf_idf_df.sort_values(by=['doc','tf_idf'],ascending=False).groupby('doc').head(15)

# In[37]: EXPORTING THE OUTPUT FILES

final.to_excel("C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\out.xlsx")

# In[39]:EXPORTING THE OUTPUT FILES

tf_idf_df.to_excel("C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\out_all.xlsx")
```

Program Three Part A: TF IDF of top 15 Bigrams words in each document.

Data Preparation: Removed the special characters and converted all words into upper case.

Output: TF*IDF values of all 15 those words.

Code Source: Provided in the blackboard. Only Modified the PhaseOne Mapper function to give bigrams as keys and built a python code to calculate the tf-idf values.

Modified Mapper Function of Phase One :

```java
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 name
String tempStr=""; //temp string to construct the key part
String term ="";
String term1="";
//loop to collect all the words
//for loop counter i is starting as we have first element of each line as document number
String prev=" ";
for(int i=1;i<docPart.length;i++)
{
prev = docPart[i-1].replaceAll("\\p{P}", ""); //removing special character and punctuation from the word

tempStr = docPart[i].replaceAll("\\p{P}", ""); //removing special character and punctuation from the word
term = prev+" "+tempStr; // Appending the previous word
term1 = term+","+docName;
word.set (term1.toUpperCase());//converting string to text writable
context.write(word,one);
}
}
}
```

Python Code to calculate the tf idf of the words:

```python
# In[2]: IMPORTING THE REQUIRED PACKAGES

import os
import pandas as pd
from io import StringIO
import numpy as np
import math

# In[3]: READING IN THE OUTPUT OF THREE MAP REDUCE PHASES

p1= pd.read_table('C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\output.txt',header=None,names=['term_doc','term_freq'])
p3= pd.read_table('C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\output3.txt',header=None,names=['word','occ'])
p2= pd.read_table('C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\output2.txt',h
eader=None,names =['doc','freq'],
converters={'doc': lambda x: str(x)})

# In[5]:CALCULATING IDF OF A TERM

idf_array =[]
for i, row in enumerate(p3.values):
idf = math.log(12/row[1])
idf_array.append(idf)
#print(idf_array)
idf_df=p3.assign(idf=idf_array)
#print(idf_df)

# In[6]:GETTING THE TOTAL NUMBER OF TERMS IN A DOCUMENT VALUE FROM PHASE2 OUTPUT

d1=dict(zip(p2['doc'],p2['freq']))
p1_1=p1.join(p1['term_doc'].str.split(',',1,expand=True).rename(columns={0:'word', 1doc'}))
p1_1['freq']=p1_1.doc.map(d1)


# In[7]: CALCULATING TF OF A TERM
tf_array=[]
for i,row in enumerate(p1_1.values):
tf=p1_1.term_freq/p1_1.freq
tf_array.append(tf)
tf_df=p1_1.assign(tf=tf_array[0])

# In[8]:GETTING THE IDF CALCULATED ABOVE

d2=dict(zip(idf_df['word'],idf_df['idf']))
tf_df['idf']=tf_df.word.map(d2)

# In[9]: CALCULATING THE FINAL TF_IDF OF WORD IN EACH DOCUMENT

tf_idf_array =[]
for i, row in enumerate(tf_df.values):
tf_idf = tf_df.tf*tf_df.idf
tf_idf_array.append(tf_idf)

tf_idf_df=tf_df.assign(tf_idf=tf_idf_array[0])

# In[39]:EXPORTING THE OUTPUT FILES

tf_idf_df.to_excel("C:\\Users\\yandr\\OneDrive\\Desktop\\Big Data\\tfidf\\out_all.xlsx”
```

Output for 3A :

Word	DOC	FREQ	DOC_FREQ	TF-Value	IDF-Value	TF*IDF Value
DUMMY TEXT	0001	2	91	0.021978022	1.791759469	0.039379329
0001 LOREM	0001	1	91	0.010989011	2.48490665	0.027306666
1500S WHEN	0001	1	91	0.010989011	1.791759469	0.019689664
1960S WITH	0001	1	91	0.010989011	1.791759469	0.019689664
A GALLEY	0001	1	91	0.010989011	1.791759469	0.019689664
A TYPE	0001	1	91	0.010989011	1.791759469	0.019689664
ALDUS PAGEMAKER	0001	1	91	0.010989011	1.791759469	0.019689664
ALSO THE	0001	1	91	0.010989011	1.791759469	0.019689664
AN UNKNOWN	0001	1	91	0.010989011	1.791759469	0.019689664
AND MORE	0001	1	91	0.010989011	1.791759469	0.019689664
AND SCRAMBLED	0001	1	91	0.010989011	1.791759469	0.019689664
AND TYPESETTING	0001	1	91	0.010989011	1.791759469	0.019689664
BEEN THE	0001	1	91	0.010989011	1.791759469	0.019689664
BOOK IT	0001	1	91	0.010989011	1.791759469	0.019689664
BUT ALSO	0001	1	91	0.010989011	1.791759469	0.019689664
45 BC	0002	2	129	0.015503876	1.791759469	0.027779217
COMES FROM	0002	2	129	0.015503876	1.791759469	0.027779217
FROM A	0002	2	129	0.015503876	1.791759469	0.027779217
0002 CONTRARY	0002	1	129	0.007751938	2.48490665	0.019262842
OF THE	0002	2	129	0.015503876	1.098612289	0.017032749
11033 OF	0002	1	129	0.007751938	1.791759469	0.013889608
2000 YEARS	0002	1	129	0.007751938	1.791759469	0.013889608
A LATIN	0002	1	129	0.007751938	1.791759469	0.013889608
A LINE	0002	1	129	0.007751938	1.791759469	0.013889608
A LOREM	0002	1	129	0.007751938	1.791759469	0.013889608
A PIECE	0002	1	129	0.007751938	1.791759469	0.013889608
A TREATISE	0002	1	129	0.007751938	1.791759469	0.013889608
AMET COMES	0002	1	129	0.007751938	1.791759469	0.013889608
AND EVIL	0002	1	129	0.007751938	1.791759469	0.013889608
AND GOING	0002	1	129	0.007751938	1.791759469	0.013889608
0003 THE	0003	1	47	0.021276596	2.48490665	0.052870354
11033 FROM	0003	1	47	0.021276596	1.791759469	0.038122542
1500S IS	0003	1	47	0.021276596	1.791759469	0.038122542
1914 TRANSLATION	0003	1	47	0.021276596	1.791759469	0.038122542
ACCOMPANIED BY	0003	1	47	0.021276596	1.791759469	0.038122542
ALSO REPRODUCED	0003	1	47	0.021276596	1.791759469	0.038122542
ARE ALSO	0003	1	47	0.021276596	1.791759469	0.038122542
BELOW FOR	0003	1	47	0.021276596	1.791759469	0.038122542
BY ENGLISH	0003	1	47	0.021276596	1.791759469	0.038122542
BY H	0003	1	47	0.021276596	1.791759469	0.038122542
CHUNK OF	0003	1	47	0.021276596	1.791759469	0.038122542
CICERO ARE	0003	1	47	0.021276596	1.791759469	0.038122542
ENGLISH VERSIONS	0003	1	47	0.021276596	1.791759469	0.038122542
EXACT ORIGINAL	0003	1	47	0.021276596	1.791759469	0.038122542

FOR THOSE	0003	1	47	0.021276596	1.791759469	0.038122542
CONTENT HERE	0004	2	104	0.019230769	1.791759469	0.034456913
0004 IT	0004	1	104	0.009615385	2.48490665	0.023893333
A LONG	0004	1	104	0.009615385	1.791759469	0.017228456
A MOREORLESS	0004	1	104	0.009615385	1.791759469	0.017228456
A PAGE	0004	1	104	0.009615385	1.791759469	0.017228456
A READER	0004	1	104	0.009615385	1.791759469	0.017228456
A SEARCH	0004	1	104	0.009615385	1.791759469	0.017228456
ACCIDENT SOMETIMES	0004	1	104	0.009615385	1.791759469	0.017228456
AND A	0004	1	104	0.009615385	1.791759469	0.017228456
AND THE	0004	1	104	0.009615385	1.791759469	0.017228456
AND WEB	0004	1	104	0.009615385	1.791759469	0.017228456
AS OPPOSED	0004	1	104	0.009615385	1.791759469	0.017228456
AS THEIR	0004	1	104	0.009615385	1.791759469	0.017228456
AT ITS	0004	1	104	0.009615385	1.791759469	0.017228456
BE DISTRACTED	0004	1	104	0.009615385	1.791759469	0.017228456
HUMOUR OR	0005	2	121	0.016528926	1.791759469	0.029615859
THE INTERNET	0005	2	121	0.016528926	1.791759469	0.029615859
0005 THERE	0005	1	121	0.008264463	2.48490665	0.020536419
INJECTED HUMOUR	0005	2	121	0.016528926	1.098612289	0.018158881
ON THE	0005	2	121	0.016528926	1.098612289	0.018158881
200 LATIN	0005	1	121	0.008264463	1.791759469	0.014807929
A DICTIONARY	0005	1	121	0.008264463	1.791759469	0.014807929
A HANDFUL	0005	1	121	0.008264463	1.791759469	0.014807929
A PASSAGE	0005	1	121	0.008264463	1.791759469	0.014807929
ALL THE	0005	1	121	0.008264463	1.791759469	0.014807929
ALTERATION IN	0005	1	121	0.008264463	1.791759469	0.014807929
ALWAYS FREE	0005	1	121	0.008264463	1.791759469	0.014807929
ANYTHING EMBARRASSING	0005	1	121	0.008264463	1.791759469	0.014807929
ARE GOING	0005	1	121	0.008264463	1.791759469	0.014807929
ARE MANY	0005	1	121	0.008264463	1.791759469	0.014807929
SIT AMET	0006	6	134	0.044776119	1.098612289	0.049191595
0006 LOREM	0006	1	134	0.007462687	2.48490665	0.018544079
AC SAPIEN	0006	1	134	0.007462687	2.48490665	0.018544079
AC VENENATIS	0006	1	134	0.007462687	2.48490665	0.018544079
ADIPISCING ELIT	0006	1	134	0.007462687	2.48490665	0.018544079
ALIQUAM CONSECTETUR	0006	1	134	0.007462687	2.48490665	0.018544079
ALIQUET ANTE	0006	1	134	0.007462687	2.48490665	0.018544079
ALIQUET NUNC	0006	1	134	0.007462687	2.48490665	0.018544079
AMET CONSECTETUR	0006	1	134	0.007462687	2.48490665	0.018544079
AMET NIBH	0006	1	134	0.007462687	2.48490665	0.018544079
AMET PELLENTESQUE	0006	1	134	0.007462687	2.48490665	0.018544079
AMET SEM	0006	1	134	0.007462687	2.48490665	0.018544079
AMET SEMPER	0006	1	134	0.007462687	2.48490665	0.018544079
AMET VESTIBULUM	0006	1	134	0.007462687	2.48490665	0.018544079
ANTE IPSUM	0006	1	134	0.007462687	2.48490665	0.018544079
0007 MAURIS	0007	1	99	0.01010101	2.48490665	0.025100067

AC PLACERAT	0007	1	99	0.01010101	2.48490665	0.025100067
ALIQUAM VARIUS	0007	1	99	0.01010101	2.48490665	0.025100067
ALIQUET FACILISIS	0007	1	99	0.01010101	2.48490665	0.025100067
AMET MI	0007	1	99	0.01010101	2.48490665	0.025100067
ANTE SED	0007	1	99	0.01010101	2.48490665	0.025100067
ARCU AC	0007	1	99	0.01010101	2.48490665	0.025100067
AUGUE AUGUE	0007	1	99	0.01010101	2.48490665	0.025100067
AUGUE NEC	0007	1	99	0.01010101	2.48490665	0.025100067
AUGUE PORTTITOR	0007	1	99	0.01010101	2.48490665	0.025100067
CONDIMENTUM SUSCIPIT	0007	1	99	0.01010101	2.48490665	0.025100067
CONGUE EU	0007	1	99	0.01010101	2.48490665	0.025100067
CRAS IMPERDIET	0007	1	99	0.01010101	2.48490665	0.025100067
CURABITUR ELEMENTUM	0007	1	99	0.01010101	2.48490665	0.025100067
DAPIBUS LACUS	0007	1	99	0.01010101	2.48490665	0.025100067
DUMMY TEXT	0008	2	91	0.021978022	1.791759469	0.039379329
0008 LOREM	0008	1	91	0.010989011	2.48490665	0.027306666
1500S WHEN	0008	1	91	0.010989011	1.791759469	0.019689664
1960S WITH	0008	1	91	0.010989011	1.791759469	0.019689664
A GALLEY	0008	1	91	0.010989011	1.791759469	0.019689664
A TYPE	0008	1	91	0.010989011	1.791759469	0.019689664
ALDUS PAGEMAKER	0008	1	91	0.010989011	1.791759469	0.019689664
ALSO THE	0008	1	91	0.010989011	1.791759469	0.019689664
AN UNKNOWN	0008	1	91	0.010989011	1.791759469	0.019689664
AND MORE	0008	1	91	0.010989011	1.791759469	0.019689664
AND SCRAMBLED	0008	1	91	0.010989011	1.791759469	0.019689664
AND TYPESETTING	0008	1	91	0.010989011	1.791759469	0.019689664
BEEN THE	0008	1	91	0.010989011	1.791759469	0.019689664
BOOK IT	0008	1	91	0.010989011	1.791759469	0.019689664
BUT ALSO	0008	1	91	0.010989011	1.791759469	0.019689664
45 BC	0009	2	129	0.015503876	1.791759469	0.027779217
COMES FROM	0009	2	129	0.015503876	1.791759469	0.027779217
FROM A	0009	2	129	0.015503876	1.791759469	0.027779217
0009 CONTRARY	0009	1	129	0.007751938	2.48490665	0.019262842
OF THE	0009	2	129	0.015503876	1.098612289	0.017032749
11033 OF	0009	1	129	0.007751938	1.791759469	0.013889608
2000 YEARS	0009	1	129	0.007751938	1.791759469	0.013889608
A LATIN	0009	1	129	0.007751938	1.791759469	0.013889608
A LINE	0009	1	129	0.007751938	1.791759469	0.013889608
A LOREM	0009	1	129	0.007751938	1.791759469	0.013889608
A PIECE	0009	1	129	0.007751938	1.791759469	0.013889608
A TREATISE	0009	1	129	0.007751938	1.791759469	0.013889608
AMET COMES	0009	1	129	0.007751938	1.791759469	0.013889608
AND EVIL	0009	1	129	0.007751938	1.791759469	0.013889608
AND GOING	0009	1	129	0.007751938	1.791759469	0.013889608
0010 THE	0010	1	47	0.021276596	2.48490665	0.052870354
11033 FROM	0010	1	47	0.021276596	1.791759469	0.038122542
1500S IS	0010	1	47	0.021276596	1.791759469	0.038122542

1914 TRANSLATION	0010	1	47	0.021276596	1.791759469	0.038122542
ACCOMPANIED BY	0010	1	47	0.021276596	1.791759469	0.038122542
ALSO REPRODUCED	0010	1	47	0.021276596	1.791759469	0.038122542
ARE ALSO	0010	1	47	0.021276596	1.791759469	0.038122542
BELOW FOR	0010	1	47	0.021276596	1.791759469	0.038122542
BY ENGLISH	0010	1	47	0.021276596	1.791759469	0.038122542
BY H	0010	1	47	0.021276596	1.791759469	0.038122542
CHUNK OF	0010	1	47	0.021276596	1.791759469	0.038122542
CICERO ARE	0010	1	47	0.021276596	1.791759469	0.038122542
ENGLISH VERSIONS	0010	1	47	0.021276596	1.791759469	0.038122542
EXACT ORIGINAL	0010	1	47	0.021276596	1.791759469	0.038122542
FOR THOSE	0010	1	47	0.021276596	1.791759469	0.038122542
CONTENT HERE	0011	2	104	0.019230769	1.791759469	0.034456913
0011 IT	0011	1	104	0.009615385	2.48490665	0.023893333
A LONG	0011	1	104	0.009615385	1.791759469	0.017228456
A MOREORLESS	0011	1	104	0.009615385	1.791759469	0.017228456
A PAGE	0011	1	104	0.009615385	1.791759469	0.017228456
A READER	0011	1	104	0.009615385	1.791759469	0.017228456
A SEARCH	0011	1	104	0.009615385	1.791759469	0.017228456
ACCIDENT SOMETIMES	0011	1	104	0.009615385	1.791759469	0.017228456
AND A	0011	1	104	0.009615385	1.791759469	0.017228456
AND THE	0011	1	104	0.009615385	1.791759469	0.017228456
AND WEB	0011	1	104	0.009615385	1.791759469	0.017228456
AS OPPOSED	0011	1	104	0.009615385	1.791759469	0.017228456
AS THEIR	0011	1	104	0.009615385	1.791759469	0.017228456
AT ITS	0011	1	104	0.009615385	1.791759469	0.017228456
BE DISTRACTED	0011	1	104	0.009615385	1.791759469	0.017228456
HUMOUR OR	0012	2	121	0.016528926	1.791759469	0.029615859
THE INTERNET	0012	2	121	0.016528926	1.791759469	0.029615859
0012 THERE	0012	1	121	0.008264463	2.48490665	0.020536419
INJECTED HUMOUR	0012	2	121	0.016528926	1.098612289	0.018158881
ON THE	0012	2	121	0.016528926	1.098612289	0.018158881
200 LATIN	0012	1	121	0.008264463	1.791759469	0.014807929
A DICTIONARY	0012	1	121	0.008264463	1.791759469	0.014807929
A HANDFUL	0012	1	121	0.008264463	1.791759469	0.014807929
A PASSAGE	0012	1	121	0.008264463	1.791759469	0.014807929
ALL THE	0012	1	121	0.008264463	1.791759469	0.014807929
ALTERATION IN	0012	1	121	0.008264463	1.791759469	0.014807929
ALWAYS FREE	0012	1	121	0.008264463	1.791759469	0.014807929
ANYTHING EMBARRASSING	0012	1	121	0.008264463	1.791759469	0.014807929
ARE GOING	0012	1	121	0.008264463	1.791759469	0.014807929
ARE GOING	0012	1	121	0.008264463	1.791759469	0.014807929

Program 3 Part B: Comparison between 2b and 3a:

2b: Unigram after removing the words that appear just once in each document.

3a: Bigram for all the words in each document.

In general, bigrams or ngrams give a better context about the document than a unigram tf-idf gives. Reason being bigrams portray more information than a unigram does. For example, **Injected Humour** is one such word that makes more sense together when characterizing the document than the individual words do. Choosing bigram signature makes more sense in this context which speaks better about the document

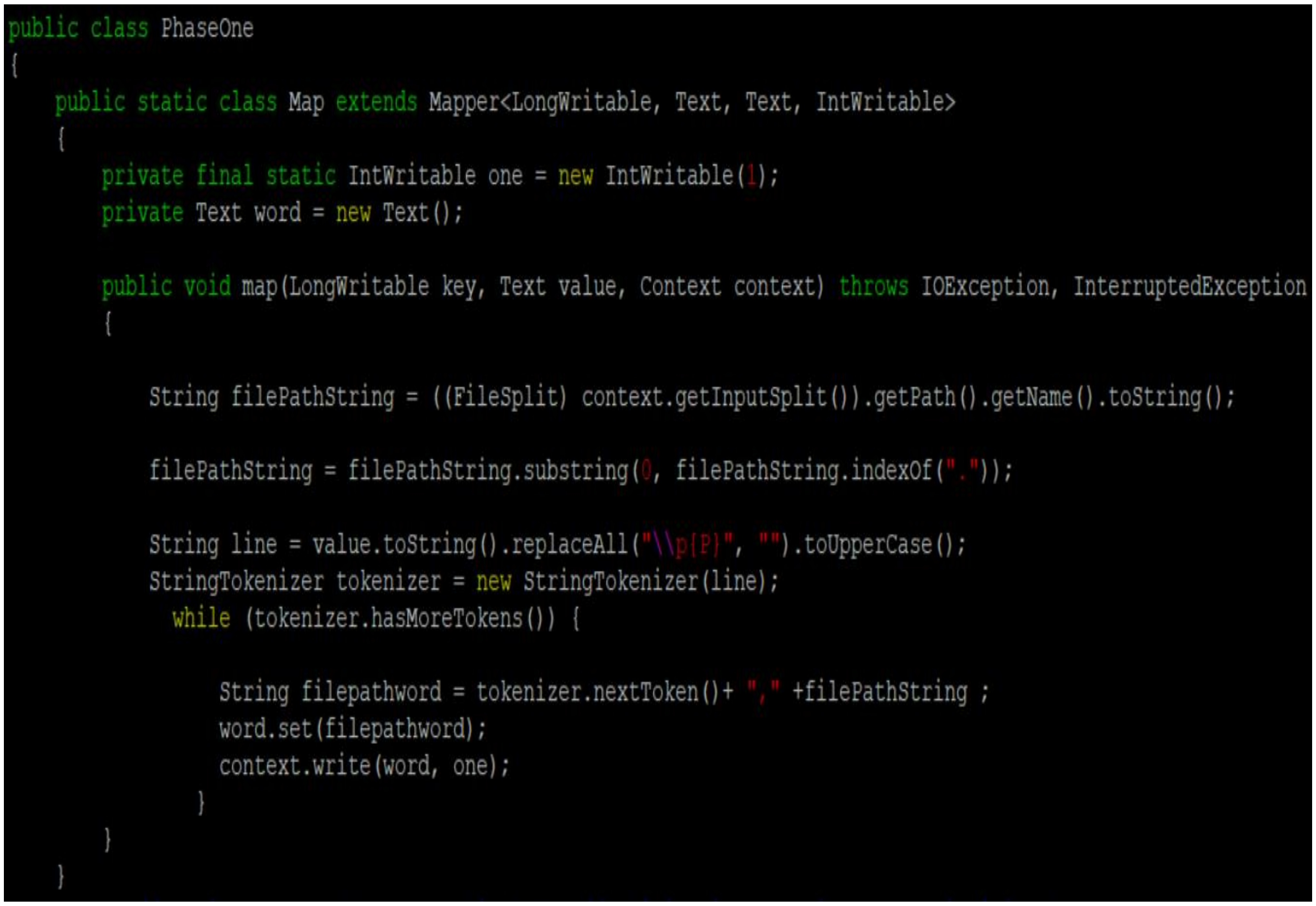
However, when the corpus is sparse, ie.,the occurrence of the bigrams together is less frequent then they might turn up having very less tf-idf value leading to be less useful than unigrams.

Program Four Part A: TF IDF of top 15 words in each document.

Data Preparation: Removed the special characters and converted all words into upper case.

Output: TF*IDF values of all 15 those words.

Code Source: Provided in the blackboard. Modified the PhaseOne Mapper function to read the document name as part of the key. Also used the same python code as of the program 3 .



```
public class PhaseOne
{
    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 filePathString = ((FileSplit) context.getInputSplit()).getPath().getName().toString();

            filePathString = filePathString.substring(0, filePathString.indexOf("."));

            String line = value.toString().replaceAll("\\p{P}", "").toUpperCase();
            StringTokenizer tokenizer = new StringTokenizer(line);
              while (tokenizer.hasMoreTokens()) {

                  String filepathword = tokenizer.nextToken()+ "," +filePathString ;
                  word.set(filepathword);
                  context.write(word, one);
              }
        }
    }
```

Output for 4A :

Word	DOC	FREQ	DOC_FREQ	TF-Value	IDF-Value	TF*IDF Value
ALICE	11-0	385	29390	0.013099694	1.386294361	0.018160032
QUEEN	11-0	68	29390	0.002313712	1.386294361	0.003207486
MOCK	11-0	56	29390	0.00190541	1.386294361	0.002641459
TURTLE	11-0	56	29390	0.00190541	1.386294361	0.002641459
GRYPHON	11-0	55	29390	0.001871385	1.386294361	0.00259429
HATTER	11-0	55	29390	0.001871385	1.386294361	0.00259429
RABBIT	11-0	43	29390	0.001463083	1.386294361	0.002028263
DORMOUSE	11-0	39	29390	0.001326982	1.386294361	0.001839588
HARE	11-0	31	29390	0.001054781	1.386294361	0.001462236
KING	11-0	61	29390	0.002075536	0.693147181	0.001438652

CATERPILLAR	11-0	27	29390	0.00091868	1.386294361	0.001273561
YOURE	11-0	23	29390	0.000782579	1.386294361	0.001084885
DUCHESS	11-0	39	29390	0.001326982	0.693147181	0.000919794
MOUSE	11-0	38	29390	0.001292957	0.693147181	0.000896209
CAT	11-0	35	29390	0.001190881	0.693147181	0.000825456
ELIZABETH	1342-0	594	124543	0.004769437	1.386294361	0.006611844
MR	1342-0	783	124543	0.006286985	0.693147181	0.004357806
DARCY	1342-0	371	124543	0.002978891	1.386294361	0.00412962
MRS	1342-0	343	124543	0.002754069	1.386294361	0.00381795
BENNET	1342-0	293	124543	0.002352601	1.386294361	0.003261398
BINGLEY	1342-0	257	124543	0.002063544	1.386294361	0.00286068
WICKHAM	1342-0	161	124543	0.001292726	1.386294361	0.001792099
COLLINS	1342-0	156	124543	0.001252579	1.386294361	0.001736444
MISS	1342-0	283	124543	0.002272308	0.693147181	0.001575044
JANE	1342-0	263	124543	0.00211172	0.693147181	0.001463733
LYDIA	1342-0	131	124543	0.001051846	1.386294361	0.001458168
CATHERINE	1342-0	110	124543	0.000883229	1.386294361	0.001224416
LIZZY	1342-0	94	124543	0.000754759	1.386294361	0.001046319
LADY	1342-0	183	124543	0.001469372	0.693147181	0.001018491
LONGBOURN	1342-0	88	124543	0.000706583	1.386294361	0.000979532
WALLPAPER	1952-0	17	9088	0.001870599	1.386294361	0.0025932
JENNIE	1952-0	12	9088	0.001320423	1.386294361	0.001830494
PATTERN	1952-0	21	9088	0.002310739	0.693147181	0.001601683
JOHN	1952-0	43	9088	0.004731514	0.287682072	0.001361172
YELLOW	1952-0	14	9088	0.001540493	0.693147181	0.001067788
DAYLIGHT	1952-0	5	9088	0.000550176	1.386294361	0.000762706
NURSERY	1952-0	5	9088	0.000550176	1.386294361	0.000762706
CREEPING	1952-0	9	9088	0.000990317	0.693147181	0.000686435
ARBORS	1952-0	4	9088	0.000440141	1.386294361	0.000610165
COLOR	1952-0	4	9088	0.000440141	1.386294361	0.000610165
CREEP	1952-0	8	9088	0.000880282	0.693147181	0.000610165
DAYTIME	1952-0	4	9088	0.000440141	1.386294361	0.000610165
GILMAN	1952-0	4	9088	0.000440141	1.386294361	0.000610165
PERKINS	1952-0	4	9088	0.000440141	1.386294361	0.000610165
PHYSICIAN	1952-0	4	9088	0.000440141	1.386294361	0.000610165
KURTZ	219-0	93	40913	0.002273116	1.386294361	0.003151208
PILGRIMS	219-0	30	40913	0.000733263	1.386294361	0.001016519
IVORY	219-0	29	40913	0.000708821	1.386294361	0.000982635
RIVER	219-0	53	40913	0.001295432	0.693147181	0.000897925
MR	219-0	51	40913	0.001246548	0.693147181	0.000864041
MEN	219-0	48	40913	0.001173221	0.693147181	0.000813215
FOREST	219-0	23	40913	0.000562169	1.386294361	0.000779331
BLACK	219-0	42	40913	0.001026569	0.693147181	0.000711563
KURTZS	219-0	21	40913	0.000513284	1.386294361	0.000711563
MANAGER	219-0	39	40913	0.000953242	0.693147181	0.000660737
STEAMER	219-0	19	40913	0.0004644	1.386294361	0.000643795
EARTH	219-0	37	40913	0.000904358	0.693147181	0.000626853

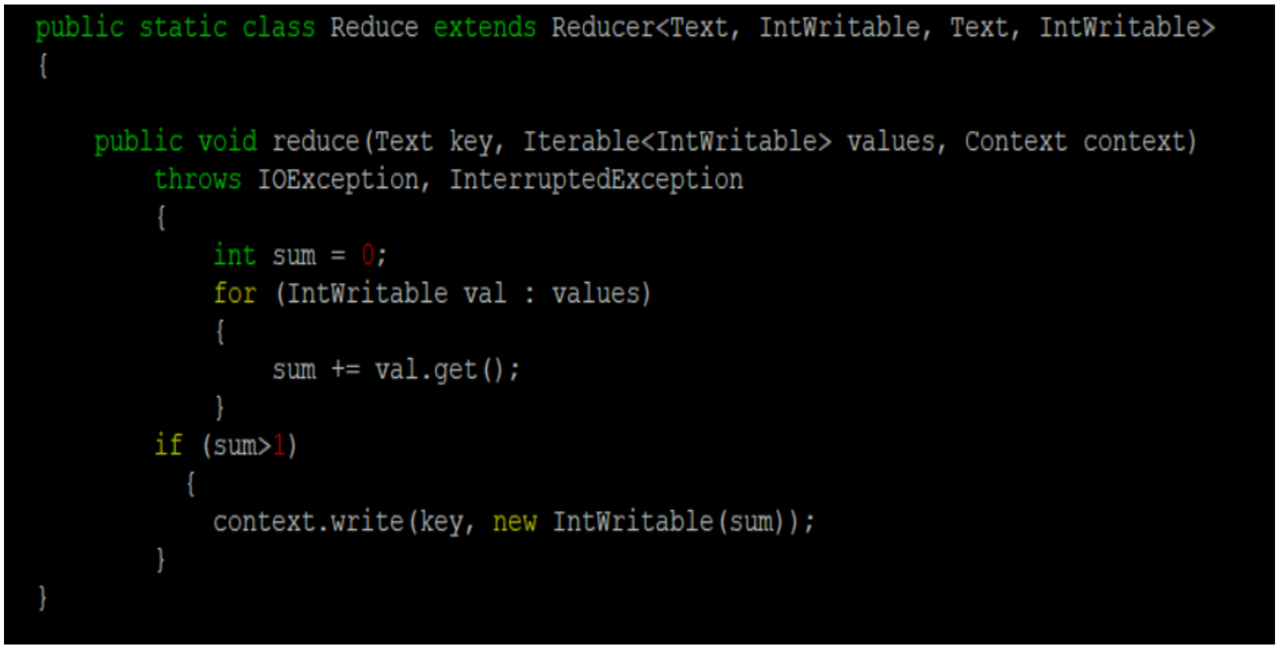
STEAMBOAT	219-0	18	40913	0.000439958	1.386294361	0.000609911
BUSH	219-0	17	40913	0.000415516	1.386294361	0.000576027
DARKNESS	219-0	31	40913	0.000757705	0.693147181	0.000525201

Program Four Part B: TF IDF of top 15 words in each document after removing the words that occur only once in each document.

Data Preparation: Removed the special characters and converted all words into upper case.

Output: TF*IDF values of all 15 those words.

Code Source: Provided in the blackboard. Modified the PhaseOne Mapper function to read the document name as part of the key(provided in 4A) and reducer phase to take ignore the words with frequency less than 2 . Also used the same python code as of the program 3 .



```
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();
            }
        if (sum>1)
          {
            context.write(key, new IntWritable(sum));
        }
}
```

Output for 4B :

Word	DOC	FREQ	DOC_FREQ	TF-Value	IDF-Value	TF*IDF Value
ALICE	11-0	385	27840	0.013829023	1.386294361	0.019171097
QUEEN	11-0	68	27840	0.002442529	1.386294361	0.003386064
MOCK	11-0	56	27840	0.002011494	1.386294361	0.002788523
TURTLE	11-0	56	27840	0.002011494	1.386294361	0.002788523
GRYPHON	11-0	55	27840	0.001975575	1.386294361	0.002738728
HATTER	11-0	55	27840	0.001975575	1.386294361	0.002738728
RABBIT	11-0	43	27840	0.00154454	1.386294361	0.002141187
DORMOUSE	11-0	39	27840	0.001400862	1.386294361	0.001942007
DUCHESS	11-0	39	27840	0.001400862	1.386294361	0.001942007
MOUSE	11-0	38	27840	0.001364943	1.386294361	0.001892212
HARE	11-0	31	27840	0.001113506	1.386294361	0.001543647
KING	11-0	61	27840	0.002191092	0.693147181	0.001518749
CATERPILLAR	11-0	27	27840	0.000969828	1.386294361	0.001344467
YOURE	11-0	23	27840	0.000826149	1.386294361	0.001145286
CAT	11-0	35	27840	0.001257184	0.693147181	0.000871413
ELIZABETH	1342-0	594	121542	0.004887199	1.386294361	0.006775097
MR	1342-0	783	121542	0.006442218	0.693147181	0.004465405

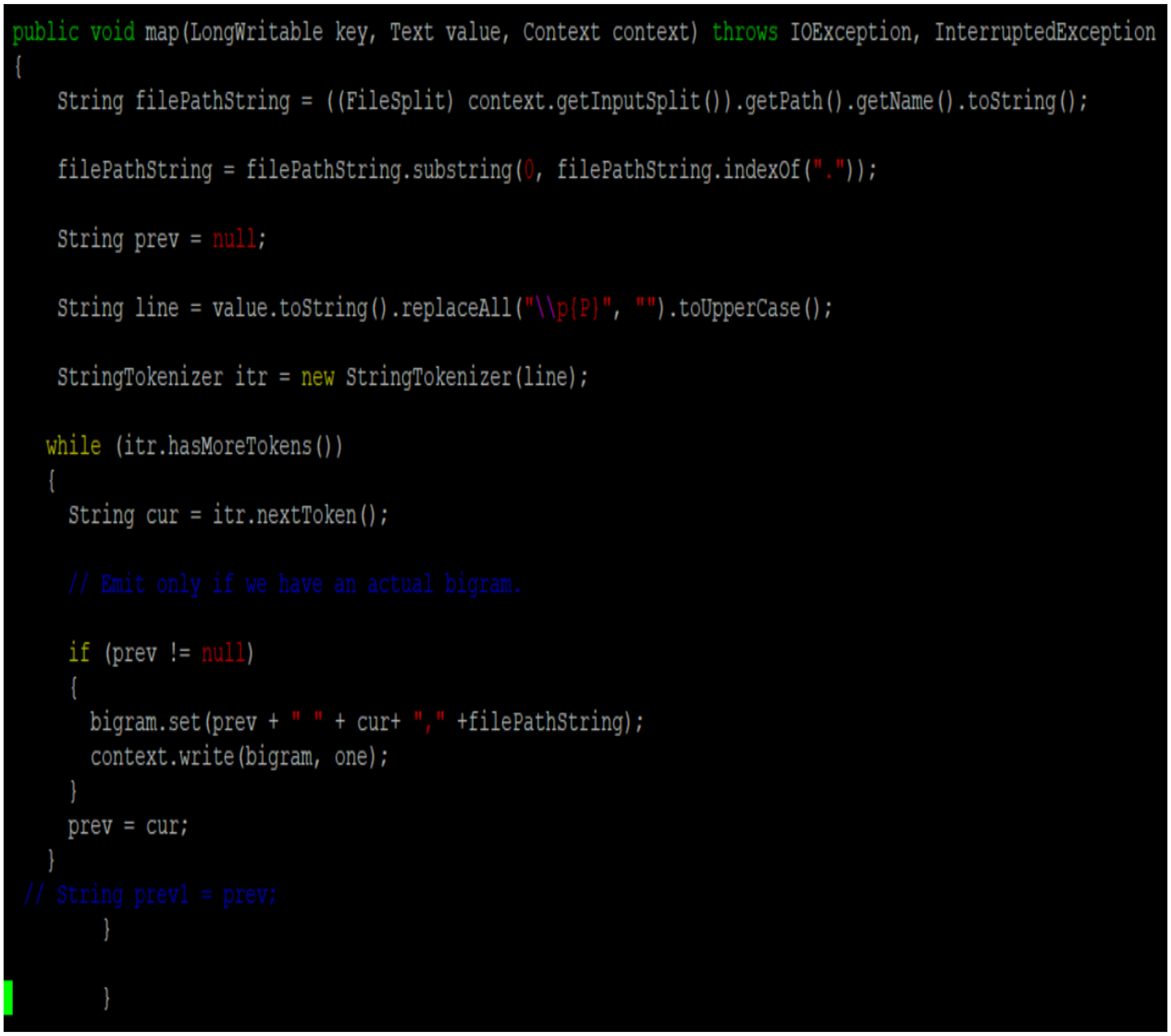
DARCY	1342-0	371	121542	0.003052443	1.386294361	0.004231584
MRS	1342-0	343	121542	0.00282207	1.386294361	0.003912219
BENNET	1342-0	293	121542	0.002410689	1.386294361	0.003341925
JANE	1342-0	263	121542	0.002163861	1.386294361	0.002999748
BINGLEY	1342-0	257	121542	0.002114495	1.386294361	0.002931313
WICKHAM	1342-0	161	121542	0.001324645	1.386294361	0.001836348
COLLINS	1342-0	156	121542	0.001283507	1.386294361	0.001779318
MISS	1342-0	283	121542	0.002328413	0.693147181	0.001613933
LYDIA	1342-0	131	121542	0.001077817	1.386294361	0.001494171
CATHERINE	1342-0	110	121542	0.000905037	1.386294361	0.001254648
LIZZY	1342-0	94	121542	0.000773395	1.386294361	0.001072153
LADY	1342-0	183	121542	0.001505652	0.693147181	0.001043639
LONGBOURN	1342-0	88	121542	0.00072403	1.386294361	0.001003718
JOHN	1952-0	43	8040	0.005348259	0.693147181	0.00370713
PATTERN	1952-0	21	8040	0.00261194	1.386294361	0.003620918
WALLPAPER	1952-0	17	8040	0.002114428	1.386294361	0.002931219
JENNIE	1952-0	12	8040	0.001492537	1.386294361	0.002069096
CREEPING	1952-0	9	8040	0.001119403	1.386294361	0.001551822
CREEP	1952-0	8	8040	0.000995025	1.386294361	0.001379397
YELLOW	1952-0	14	8040	0.001741294	0.693147181	0.001206973
DAYLIGHT	1952-0	5	8040	0.000621891	1.386294361	0.000862123
NURSERY	1952-0	5	8040	0.000621891	1.386294361	0.000862123
ARBORS	1952-0	4	8040	0.000497512	1.386294361	0.000689699
COLOR	1952-0	4	8040	0.000497512	1.386294361	0.000689699
DARLING	1952-0	4	8040	0.000497512	1.386294361	0.000689699
DAYTIME	1952-0	4	8040	0.000497512	1.386294361	0.000689699
GILMAN	1952-0	4	8040	0.000497512	1.386294361	0.000689699
PERKINS	1952-0	4	8040	0.000497512	1.386294361	0.000689699
KURTZ	219-0	93	37253	0.002496443	1.386294361	0.003460805
BLACK	219-0	42	37253	0.001127426	1.386294361	0.001562944
MANAGER	219-0	39	37253	0.001046896	1.386294361	0.001451305
DARKNESS	219-0	31	37253	0.000832148	1.386294361	0.001153602
PILGRIMS	219-0	30	37253	0.000805304	1.386294361	0.001116389
IVORY	219-0	29	37253	0.000778461	1.386294361	0.001079176
RIVER	219-0	53	37253	0.001422704	0.693147181	0.000986143
MR	219-0	51	37253	0.001369017	0.693147181	0.00094893
MEN	219-0	48	37253	0.001288487	0.693147181	0.000893111
FOREST	219-0	23	37253	0.0006174	1.386294361	0.000855898
STATION	219-0	45	37253	0.001207956	0.693147181	0.000837292
KURTZS	219-0	21	37253	0.000563713	1.386294361	0.000781472
WILDERNESS	219-0	21	37253	0.000563713	1.386294361	0.000781472
STEAMER	219-0	19	37253	0.000510026	1.386294361	0.000707046
EARTH	219-0	37	37253	0.000993209	0.693147181	0.00068844

Program Four Part C: TF IDF of top 15 Bigram words in each document.

Data Preparation: Removed the special characters and converted all words into upper case.

Output: TF*IDF values of all 15 those words.

Code Source: Provided in the blackboard. Modified the PhaseOne Mapper function to read the document name as part of the key and to get the bigrams. Also, used the same python code as of the program 3 .



```
public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException
{
    String filePathString = ((FileSplit) context.getInputSplit()).getPath().getName().toString();

    filePathString = filePathString.substring(0, filePathString.indexOf("."));

    String prev = null;

    String line = value.toString().replaceAll("\\p{P}", "").toUpperCase();

    StringTokenizer itr = new StringTokenizer(line);

   while (itr.hasMoreTokens())
   {
     String cur = itr.nextToken();

     // Emit only if we have an actual bigram.

     if (prev != null)
     {
       bigram.set(prev + " " + cur+ "," +filePathString);
       context.write(bigram, one);
     }
     prev = cur;
   }
  // String prev1 = prev;
         }

         }
```

Output for 4C :

Word	DOC	FREQ	DOC_FREQ	TF-Value	IDF-Value	TF*IDF Value
SAID ALICE	11-0	111	26608	0.004171678	1.386294361	0.005783173
THE QUEEN	11-0	60	26608	0.002254961	1.386294361	0.00312604
THE MOCK	11-0	53	26608	0.001991882	1.386294361	0.002761335
MOCK TURTLE	11-0	51	26608	0.001916717	1.386294361	0.002657134
THE GRYPHON	11-0	50	26608	0.001879134	1.386294361	0.002605033

THE HATTER	11-0	50	26608	0.001879134	1.386294361	0.002605033
SAID THE	11-0	208	26608	0.007817198	0.287682072	0.002248868
THE DUCHESS	11-0	37	26608	0.001390559	1.386294361	0.001927724
THE DORMOUSE	11-0	34	26608	0.001277811	1.386294361	0.001771422
MARCH HARE	11-0	30	26608	0.00112748	1.386294361	0.00156302
THE KING	11-0	57	26608	0.002142213	0.693147181	0.001484869
THE MARCH	11-0	28	26608	0.001052315	1.386294361	0.001458818
THE CATERPILLAR	11-0	25	26608	0.000939567	1.386294361	0.001302516
THE MOUSE	11-0	25	26608	0.000939567	1.386294361	0.001302516
THOUGHT ALICE	11-0	25	26608	0.000939567	1.386294361	0.001302516
MR DARCY	1342-0	228	113516	0.002008527	1.386294361	0.00278441
MRS BENNET	1342-0	123	113516	0.001083548	1.386294361	0.001502116
MR COLLINS	1342-0	121	113516	0.001065929	1.386294361	0.001477691
LADY CATHERINE	1342-0	87	113516	0.000766412	1.386294361	0.001062472
MR BINGLEY	1342-0	83	113516	0.000731174	1.386294361	0.001013623
MR BENNET	1342-0	75	113516	0.0006607	1.386294361	0.000915924
MISS BINGLEY	1342-0	67	113516	0.000590225	1.386294361	0.000818226
SHE COULD	1342-0	123	113516	0.001083548	0.693147181	0.000751058
ELIZABETH WAS	1342-0	61	113516	0.000537369	1.386294361	0.000744952
AM SURE	1342-0	58	113516	0.000510941	1.386294361	0.000708315
MR WICKHAM	1342-0	55	113516	0.000484513	1.386294361	0.000671678
HER MOTHER	1342-0	52	113516	0.000458085	1.386294361	0.000635041
MISS BENNET	1342-0	52	113516	0.000458085	1.386294361	0.000635041
OF HER	1342-0	244	113516	0.002149477	0.287682072	0.000618366
MRS GARDINER	1342-0	50	113516	0.000440467	1.386294361	0.000610616
JOHN IS	1952-0	9	8222	0.001094624	1.386294361	0.001517471
JOHN SAYS	1952-0	6	8222	0.000729749	1.386294361	0.001011648
YELLOW WALLPAPER	1952-0	6	8222	0.000729749	1.386294361	0.001011648
BY DAYLIGHT	1952-0	5	8222	0.000608125	1.386294361	0.00084304
THE PAPER	1952-0	9	8222	0.001094624	0.693147181	0.000758736
A PHYSICIAN	1952-0	4	8222	0.0004865	1.386294361	0.000674432
AROUND THE	1952-0	4	8222	0.0004865	1.386294361	0.000674432
CHARLOTTE PERKINS	1952-0	4	8222	0.0004865	1.386294361	0.000674432
JOHN AND	1952-0	4	8222	0.0004865	1.386294361	0.000674432
JOHN WOULD	1952-0	4	8222	0.0004865	1.386294361	0.000674432
PERKINS GILMAN	1952-0	4	8222	0.0004865	1.386294361	0.000674432
THE DAYTIME	1952-0	4	8222	0.0004865	1.386294361	0.000674432
THE PATTERN	1952-0	8	8222	0.000972999	0.693147181	0.000674432
THE WALLPAPER	1952-0	4	8222	0.0004865	1.386294361	0.000674432
IS ONE	1952-0	7	8222	0.000851374	0.693147181	0.000590128
MR KURTZ	219-0	39	37506	0.001039834	1.386294361	0.001441515
AS THOUGH	219-0	34	37506	0.000906522	1.386294361	0.001256706
THE MANAGER	219-0	33	37506	0.000879859	1.386294361	0.001219744
I SAW	219-0	40	37506	0.001066496	0.693147181	0.000739239
THE PILGRIMS	219-0	19	37506	0.000506586	1.386294361	0.000702277
THE FOREST	219-0	16	37506	0.000426598	1.386294361	0.000591391
HE WAS	219-0	74	37506	0.001973018	0.287682072	0.000567602

OF DARKNESS	219-0	14	37506	0.000373274	1.386294361	0.000517467
THE STATION	219-0	14	37506	0.000373274	1.386294361	0.000517467
THE WILDERNESS	219-0	14	37506	0.000373274	1.386294361	0.000517467
KURTZ WAS	219-0	13	37506	0.000346611	1.386294361	0.000480505
THE STEAMER	219-0	13	37506	0.000346611	1.386294361	0.000480505
OF HIS	219-0	58	37506	0.001546419	0.287682072	0.000444877
THE RIVER	219-0	24	37506	0.000639898	0.693147181	0.000443543
KURTZ HAD	219-0	10	37506	0.000266624	1.386294361	0.000369619

Program FOUR PART D: Comments on the PART A, PART B and PART C of the program 4.

PART A – Unigrams without filtering	**PART B – Unigrams after filtering less frequent words**	**PART C – Bigrams without filtering**
The top 15 words in each document form the signature of the document.	Top 15 words of each document with increased TF*IDF values form the signature of the document. Better than the Part A output.	Top 15 bigrams of each document form the signature of the document. Better than the unigrams as the corpse is not sparse.

When the output for Bigrams is observed, we note that the corpse is not a sparse one, that is, the occurrence of bigrams is a significant number and hence a bigram signature in this context best represents the document rather than a unigram signature.

For instance, it is very fascinating to see how the bigrams of the document 1342-0 listed all the characters that are present in the story of the document and whether they are male or female as they have the titles associated (MR or MRS).

Also please refer this link for “ tf idf” values of all the words :
https://drive.google.com/drive/u/0/folders/1GPyACEdTFq52ZJ1UCfqz3sCJSEFjWi77