Program One: Frequencies of Words and their Frequencies. (Only words greater than 5 frequency)

Data Preparation: Removed Special characters and converted all to lower case.

Output: First and Last 50 words.

```
// importing the Required Libraries -- Remain same for first and second programs

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class first_program {

// My Mapper Class

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

// Passing the string of tokens to the mapper as (key, value) pair

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {

     // Filtering the data for alpha numerics and then converting them to lower case.

      String line = value.toString().replaceAll("[^\\p{L}\\p{Z}]","").toLowerCase();
      StringTokenizer itr = new StringTokenizer(line);
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }

//My Reducer Class

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

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

```java
      }
    // Filtering for words with frequency greater than 5
      if(sum>5) {
      result.set(sum);
      context.write(key, result);
    }
   }
  }

//My Driver Section

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(first_program.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

First 50 words and their Frequencies:

a	34572
aaron	30
abandon	8
abandond	6
abandoned	17
abbey	9
abhorred	12
abhorrence	12
abilities	23
ability	41
able	368
aboard	49
abode	25
abodes	27
abolished	7
abominable	6
abounding	7
about	2455
above	414
abraham	46
abroad	37
absence	60
absent	32
absolute	73
absolutely	79
absolution	7
absolved	6

absorbed	7
absurd	45
absurdities	9
absurdity	12
abuse	15
acamas	8
accents	12
accept	57
acceptable	6
acceptation	6
accepted	48
accepteth	7
access	224
accident	43
accidents	28
acclaim	6
accompanied	21
accompany	6
accomplished	28
accord	6
according	161
accordingly	31
account	258

Last 50 words and their Frequencies:

here	12
his	6
how	10
i	571
if	46
ill	23
im	13
in	16
is	15
it	120
its	32
ive	9
jim	42
just	9
look	8
my	13
next	6
no	22
now	34
of	9
oh	8
one	14
pap	6
pretty	15
say	6
says	9
she	61

so	146
some	10
sometimes	8
that	37
thats	15
the	141
then	103
there	64
these	6
they	75
this	13
tom	21
we	111
well	87
what	29
whats	11
when	43
where	10
who	6
why	63
yes	13
you	98
your	7
	14
æschere	6
þær	8
ćneas	58
ćtolian	6

Program Two: Frequencies of Bigrams and their Frequencies. (Only 2-words greater than 5 frequency). Driver class and Libraries are same as First Program.

Data Preparation: Removed Special characters.

Output: First and Last 50 2-words.

```
// Mapper Class

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private static final Text bigram = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {

// Filtering the text for any special characters only

           String line = value.toString().replaceAll("[^\\p{L}\\p{Z}]","");
           String prev = null;

           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);
               context.write(bigram, one);
             }
             prev = cur;
           }
          String prev1 = prev; // concatenating the first word of a line with the last word of previous line
   }

}

// Reducer Class

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

    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
  // Filtering the words with frequency greater than 5

   if (sum >5){

      result.set(sum);
      context.write(key, result);
    }

    }

  }

// Driver Class

public class second {

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

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(second.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);

  }
}
```

First 50 Bigrams and their Frequencies :

A Commonwealth	9
A Law	6
A Man	7
A chief	6
A man	6
A moment	6
A mortal	6
A sudden	9
ADVENTURE OF	6
AND OF	6
AND THE	18
Above the	9
According to	6
Achilles and	14
Achilles arms	6
Achilles hand	7
Achilles in	6
Achilles thus	8
Achilles to	9
Achilles with	6
Actions and	6
Acts of	11
Administration of	8
Adventure of	6
After a	51
After all	6
After this	6

Again I	6
Again he	6
Again the	9
Against The	7
Against the	11
Ah said	16
Ah yes	6
Ahab and	19
Ahab did	7
Ahab had	12
Ahab in	9
Ahab is	11
Ahab now	8
Ahab said	6
Ahab stood	11
Ahab the	7
Ahab to	9
Ahab was	16
Ahab who	8
Ahab with	6
Ahab would	6
Ajax and	8
Ajax the	8

Last 50 Bigrams and their Frequencies:

I reckoned	9
I said	13
I says	35
I see	11
I set	6
I thought	6
I told	6
I took	8
I was	41
I went	15
I wish	6
If I	7
If he	6
If you	9
It aint	6
It warnt	8
It was	50
Its a	7
Its the	7
Jim said	13
Of course	8
Pretty soon	12
She said	9
She says	6
She was	11
So I	40
So he	7
So the	9

So then	12
So we	14
That was	12
The duke	7
The king	13
The old	8
Then I	20
Then he	20
Then the	15
Then we	12
There aint	10
There warnt	11
There was	33
They all	7
We got	6
We was	6
We went	6
Well I	19
Well then	6
When I	11
When we	11
You see	6

Program Three: 100 Most Frequent words. 2 phases of map and reduce used where the second mapper swaps the key-value pairs and passes to the reducer. Comparator to get the output in descending order.

Data Preparation: Filtered for only words.

```java
// Libraries - Additional are the Job control libraries

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;
import org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;
import org.apache.hadoop.mapreduce.lib.jobcontrol.JobControl;

// First Mapper Class

public class WordMapper extends Mapper<LongWritable, Text, Text, IntWritable> {

 IntWritable intWritable = new IntWritable(1);
 Text text = new Text();

 @Override
 public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

// Filtering the text without numbers or spcial characters

  String line = value.toString().replaceAll("[^\\p{L}\\p{Z}]","");

  for (String word : line.split("\\W+")) {
    if (word.length() > 0) {
      text.set(word);
      context.write(text, intWritable);
     }
   }
 }
}

// First Reducer Class

  public class SumReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

    IntWritable intWritable = new IntWritable();

    @Override
    public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

      int wordCount = 0;
      for (IntWritable value : values) {
```

```
                wordCount += value.get();
            }
            intWritable.set(wordCount);
            context.write(key, intWritable);
          }
        }

// Second Mapper Class -- Swaps the output of first reducer

    public class WordMapper2 extends Mapper< Text, Text, IntWritable, Text> {

          IntWritable frequency = new IntWritable();

          @Override
          public void map(Text key, Text value, Context context)
            throws IOException, InterruptedException {

            int newVal = Integer.parseInt(value.toString());
            frequency.set(newVal);
            context.write(frequency, key);
          }
        }

// Second reducer CLass

    public class SumReducer2 extends Reducer<IntWritable, Text, IntWritable, Text> {

          Text word = new Text();

          @Override
          public void reduce(IntWritable key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException {

            for (Text value : values) {
                word.set(value);
                context.write(key, word);
            }
          }
        }

 // Comparator Class – For Descending Order.

package stubs;
import java.nio.ByteBuffer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.WritableComparator;

public class IntComparator extends WritableComparator {

        public IntComparator() {
          super(IntWritable.class);
        }

        @Override
        public int compare(byte[] b1, int s1, int l1, byte[] b2,
              int s2, int l2) {
          Integer v1 = ByteBuffer.wrap(b1, s1, l1).getInt();
          Integer v2 = ByteBuffer.wrap(b2, s2, l2).getInt();
          return v1.compareTo(v2) * (-1);
        }
      }
```

```java
// Driver Class

public class WordCombined extends Configured implements Tool {

 public int run(String[] args) throws Exception {

    JobControl jobControl = new JobControl("jobChain");
    Configuration conf1 = getConf();

    Job job1 = Job.getInstance(conf1);
    job1.setJarByClass(WordCombined.class);
    job1.setJobName("Word Combined");

    FileInputFormat.setInputPaths(job1, new Path(args[0]));
    FileOutputFormat.setOutputPath(job1, new Path(args[1] + "/temp"));

//Job 1
    job1.setMapperClass(WordMapper.class);
    job1.setReducerClass(SumReducer.class);
    job1.setCombinerClass(SumReducer.class);

    job1.setOutputKeyClass(Text.class);
    job1.setOutputValueClass(IntWritable.class);

    ControlledJob controlledJob1 = new ControlledJob(conf1);
    controlledJob1.setJob(job1);

    jobControl.addJob(controlledJob1);
    Configuration conf2 = getConf();

//Job2

    Job job2 = Job.getInstance(conf2);
    job2.setJarByClass(WordCombined.class);
    job2.setJobName("Word Invert");

    FileInputFormat.setInputPaths(job2, new Path(args[1] + "/temp"));
    FileOutputFormat.setOutputPath(job2, new Path(args[1] + "/final"));

    job2.setMapperClass(WordMapper2.class);
    job2.setReducerClass(SumReducer2.class);
    job2.setCombinerClass(SumReducer2.class);

    job2.setOutputKeyClass(IntWritable.class);
    job2.setOutputValueClass(Text.class);
    job2.setInputFormatClass(KeyValueTextInputFormat.class);

    job2.setSortComparatorClass(IntComparator.class);
    ControlledJob controlledJob2 = new ControlledJob(conf2);
    controlledJob2.setJob(job2);

    // make job2 dependent on job1
    controlledJob2.addDependingJob(controlledJob1);

    // add the job to the job control
    jobControl.addJob(controlledJob2);
    Thread jobControlThread = new Thread(jobControl);
    jobControlThread.start();

// Printing the Job States

```

```java
       while (!jobControl.allFinished()) {
       System.out.println("Jobs in waiting state: " + jobControl.getWaitingJobList().size());
           System.out.println("Jobs in ready state: " + jobControl.getReadyJobsList().size());
           System.out.println("Jobs in running state: " + jobControl.getRunningJobList().size());
           System.out.println("Jobs in success state: " + jobControl.getSuccessfulJobList().size());
           System.out.println("Jobs in failed state: " + jobControl.getFailedJobList().size());
       try {
           Thread.sleep(5000);
           } catch (Exception e) {

           }

         }
          System.exit(0);
          return (job1.waitForCompletion(true) ? 0 : 1);
         }
         public static void main(String[] args) throws Exception {
         int exitCode = ToolRunner.run(new WordCombined(), args);
         System.exit(exitCode);
         }
       }

```

100 Most Frequent Words with their Frequencies:

99423	the
58906	and
52249	of
47666	to
33190	a
28356	in
27772	I
23083	that
18836	it
18387	was
18232	his
17844	he
14776	is
14173	with
12982	as
12369	for
12031	you
11447	not
10698	be
10106	had
9407	s
9397	on
9385	by
9168	but
9107	The
9042	him
9035	all

8924	at
8362	her
7943	have
7937	my
7568	from
7510	or
7416	they
7202	this
7117	which
7001	me
6873	so
5915	she
5798	said
5641	are
5489	their
5484	And
5461	one
5329	were
5238	them
5139	no
5102	there
4774	we
4755	when
4392	out
4361	t
4090	an
4080	would
4077	if
3986	up

3984	d
3821	He
3819	But
3777	been
3742	what
3693	will
3623	any
3566	man
3540	more
3517	then
3419	could
3306	into
3283	some
3280	do
3258	who
3166	other
3140	It
3119	your
2968	now
2952	time
2838	can
2781	very
2668	such
2643	upon
2625	may
2567	down
2559	see
2505	like
2500	than

```
2450    before
2419    shall
2401    our
2388    little
2375    about
2360    must
2306    has
2252    know
2238    did
2227    over
2215    Mr
2170    only
2081    should
2070    men
2033    again
```

References:

https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

https://stackoverflow.com/questions/2499585/chaining-multiple-mapreduce-jobs-in-hadoop

https://coe4bd.github.io/HadoopHowTo/multipleJobsSingle/multipleJobsSingle.html

COMMANDS USED :

Compiling Java file:

```
$hadoop com.sun.tools.javac.Main <filename.java> -d <destination directory>
```

Creating Jar file:

```
$jar -cvf <filename.jar> -C <destination directory> <target directory>
```

Putting file to hdfs:

```
$hadoop fs -put <filename> <destination directory on hdfs>
```

Running job on hadoop:

```
hadoop jar <filename.jar> <filename without extension> <input data location on hdfs> <output data location on hdfs>
```

Output of first 50 words :

```
hadoop          fs -cat <output_file> | head -50 >  output_to local_file
```

Output of last 50 words :

```
hadoop          fs -cat <output_file> | head -50 >  output_to local_file
```