DIC - HW2 - MAP REDUCE

[redacted]

Contents

For this homework assignment, we will delve into Hadoop, giving you the opportunity to gain hands-on experience in writing and executing MapReduce code. Additionally, we will conduct subsequent analysis. In broad terms, the tasks involve the installation and configuration of Hadoop and MapReduce, the creation of a simple word-count program, and responding to inquiries related to data flow within a MapReduce application.

About My Environment

- **Hadoop Version**: Hadoop 3.2.1
- **Source Code Repository**: https://gitbox.apache.org/repos/asf/hadoop.git
    - Commit Reference: b3cbbb467e22ea829b3808f4b7b01d07e0bf3842

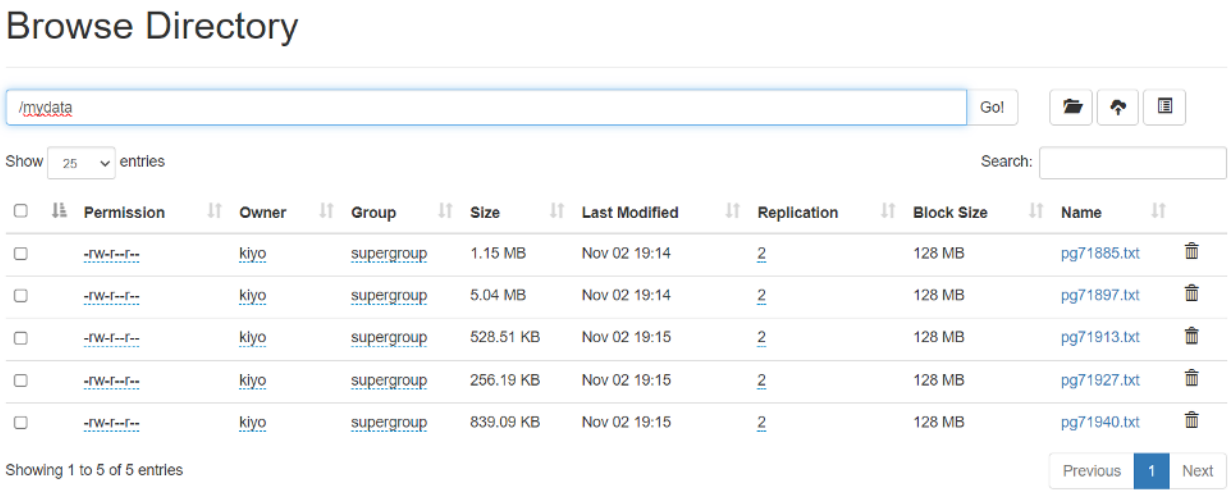
- **Compilation Environment**:
  - Compiled with Protoc Version: 2.5.0
  - Compilation Command Location: /usr/local/hadoop/share/hadoop/common/hadoop-common-3.2.1.jar
  - Environment: Windows Subsystem for Linux (WSL) - Ubuntu 20.04
- **Java Runtime Environment**:
  - OpenJDK Version: 1.8.0_382
  - OpenJDK Runtime Environment: build 1.8.0_382-8u382-ga-1~22.04.1-b05
  - Java Virtual Machine: 64-Bit Server VM, build 25.382-b05, mixed mode

```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ hadoop version
Hadoop 3.2.1
Source code repository https://gitbox.apache.org/repos/asf/hadoop.git -r b3cbbb467e22ea829b3808f4b7b01d07e0bf3842
Compiled by rohithsharmaks on 2019-09-10T15:56Z
Compiled with protoc 2.5.0
From source with checksum 776eaf9eee9c0ffc370bcbc1888737
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.2.1.jar
```

Basic Word Count

Before we proceed to count the words, as I mentioned in above I've placed my Hadoop setup in the WSL, which means i need to transfer the files of the books I downloaded from the Project Gutenberg to Hadoop. The command goes like `hadoop fs -put <input-file-path> <destination-path>`

The below figure shows the five books in the Hadoop placed in the `mydata` directory in text format that was downloaded from the project Gutenberg each is more than 100kb which satisfies the requirements of the task.



Now let's focus on counting the words with the most frequency from the whole book, though thre's a demo program that's given MapReduce tutorial documentation that program fails to remove the punctuation so it's modified to satisfy the following cases, that list should not include any punctuation (other than punctuation that is actually part of the word), and should be case insensitive (Dog and dog would be counted as the same word). Let's have a look at the below code for better understanding.

Firstly we import the required libraries inorder to perform the task, these are important for the Hadoop and Map reduce as they give access to the Hadoop libraries and classes that are essential to implement the Map reduce tasks.

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

Next define the `TokenizerMapper` class, which extends the `Mapper` class provided by Hadoop.

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

In the `map` method within the `TokenizerMapper` class, which is executed for each input key-value pair.

```
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
    String token = itr.nextToken().replaceAll("[^a-zA-Z]", "").toLowerCase();
    if (!token.isEmpty()) {
        word.set(token);
        context.write(word, one);
    }
}
```

This is the crucial part of the code where the modification happened to filter out the dupicates and characters like ‘ / ‘ , “ , etc by utilizing regrex `replaceAll("[^a-zA-Z]", "")` to remove any non-alphabetical characters (punctuation) from the word by doing this we’re only extracting the alphebetx from the texts, and now Convert the word to lowercase using `.toLowerCase()`. This will help to satisfy the requirements to not include duplicates and punctuation in the counts.

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

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

Here the `reduce` method within the `IntSumReducer` class, which is executed for each unique word which contains the values of the counts for each words.

```
int sum = 0;
for (IntWritable val : values) {
    sum += val.get();
}
```

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

Here in the reduce method the code iterates through the values, which are the counts of the words emitted by mappers, within the `reduce` method. It calculates the total word count by adding the separate counts. It returns the ‘ result' as a `IntWritable` with the overall count. Finally, it uses context.write (key, result) to print the word (key) with its overall count.

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.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);
}
```

- The `main` method is the entry point of the program. This calls for the Map Reducer to start by setting up Hadoop `Configuration and` sets various properties.
- It then sets the various classes like mapper, reducer, output key and value classes.
- After that it specifies the input and output paths. then it waits for the job to complete and exits the program.

Now lets have a look at the whole code:

```
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;
```

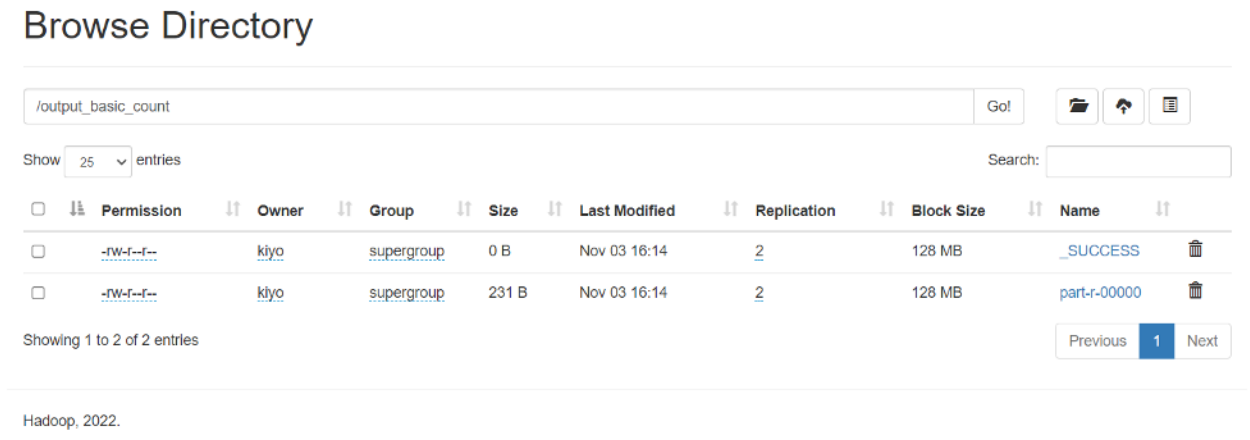
```
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {

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

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

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        String token = itr.nextToken().replaceAll("[^a-zA-Z]", "").toLowerCase();  // Remo
ve punctuation and make it lowercase
        if (!token.isEmpty()) {
          word.set(token);
          context.write(word, one);
        }
      }
    }
  }

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

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.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);
```

```
    }
}
```

Initial Examination

First compile it `javac -classpath hadoop classpath -d /mnt/c/users/tarun/desktop/dic/hw2/basic_count/` `WordCount.java` the create a .jar file `jar cf wc.jar WordCount*.class` , finally save the output to a folder `hadoop jar 'wc.jar' WordCount /mydata /basic_output`

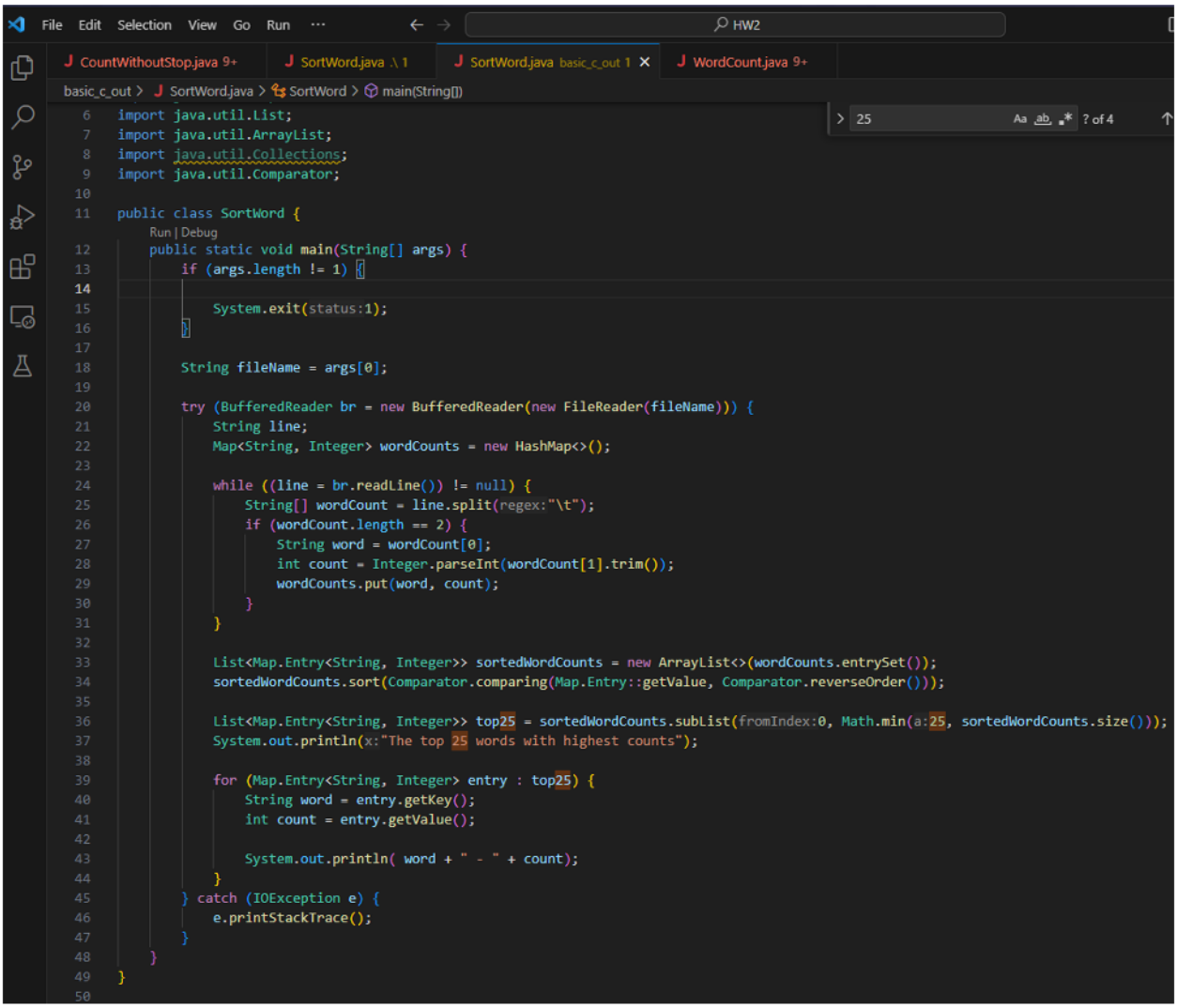
```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ javac -classpath `hadoop classpath` -d /mnt/c/users/tarun/desktop/dic/hw2/b
asic_count/ WordCount.java
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ cd basic_count/
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/basic_count$ jar cf wc.jar WordCount*.class
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/basic_count$ hadoop jar  'uc.jar' WordCount /mydata /basic_output
JAR does not exist or is not a normal file: /mnt/c/users/tarun/desktop/dic/hw2/basic_count/uc.jar
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/basic_count$ hadoop jar  'wc.jar' WordCount /mydata /basic_output
2023-11-04 20:16:14,637 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2023-11-04 20:16:15,134 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
2023-11-04 20:16:15,519 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool in
terface and execute your application with ToolRunner to remedy this.
2023-11-04 20:16:15,538 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/kiyo/.stagi
ng/job_1698962573900_0018
2023-11-04 20:16:15,641 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
2023-11-04 20:16:15,759 INFO input.FileInputFormat: Total input files to process : 5
2023-11-04 20:16:15,790 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
2023-11-04 20:16:15,810 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
2023-11-04 20:16:15,824 INFO mapreduce.JobSubmitter: number of splits:5
2023-11-04 20:16:15,955 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
```



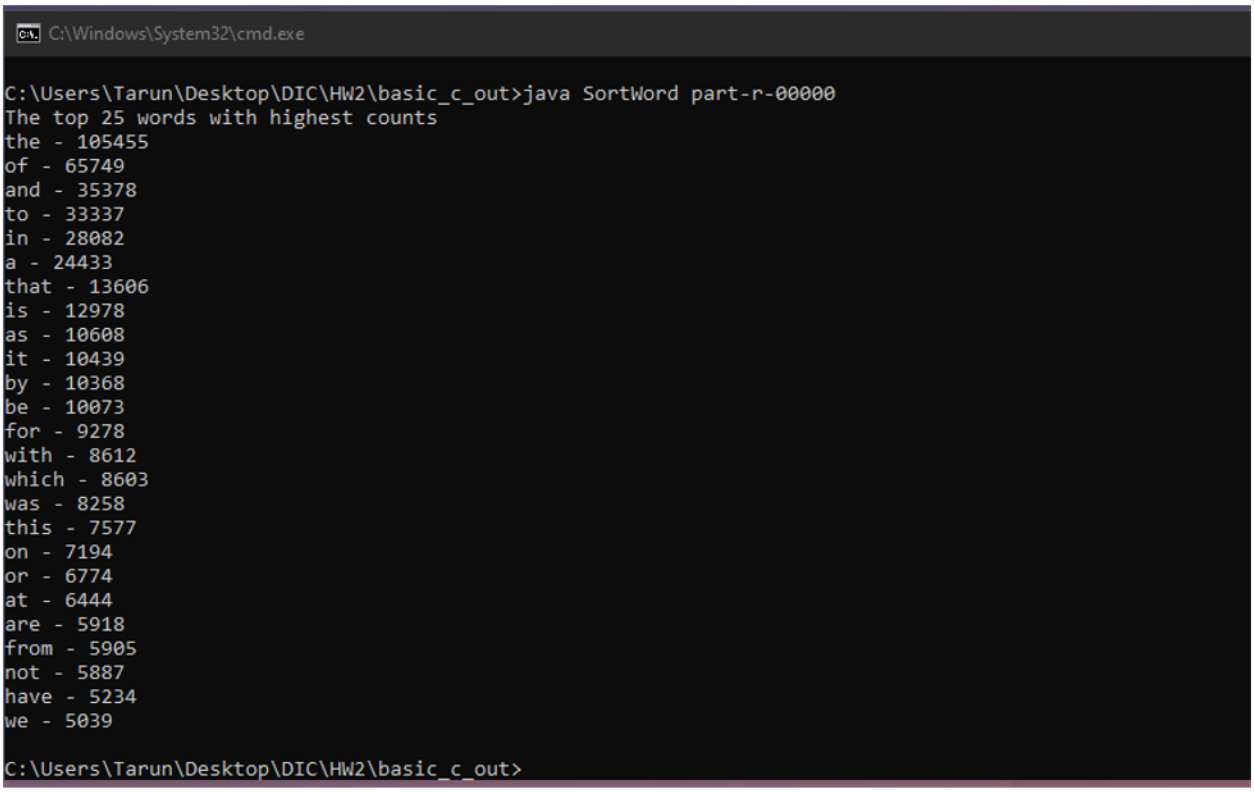
Now let's get the output files form the Hadoop to local `hadoop fs -copyToLocal /basic_output/* /mnt/c/users/tarun/desktop/dic/hw2`

```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp$ hadoop fs -copyToLocal /basic_output/* /mnt/c/users/tarun/deskt
op/dic/hw2
2023-11-04 20:45:42,590 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2023-11-04 20:45:43,296 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
```

As you can see form the above picture theese words are sorted based on the key values, so we need to sosort them based on the count using a simple java program. as shown below.



This code takes the filename as input and print the top 25 words with most freequency as the output. Now let's have look into the output.



As you can see there are soo many stop words, so in the next iteration we'll exclude them and see the difference.

Basic Word Count Extended

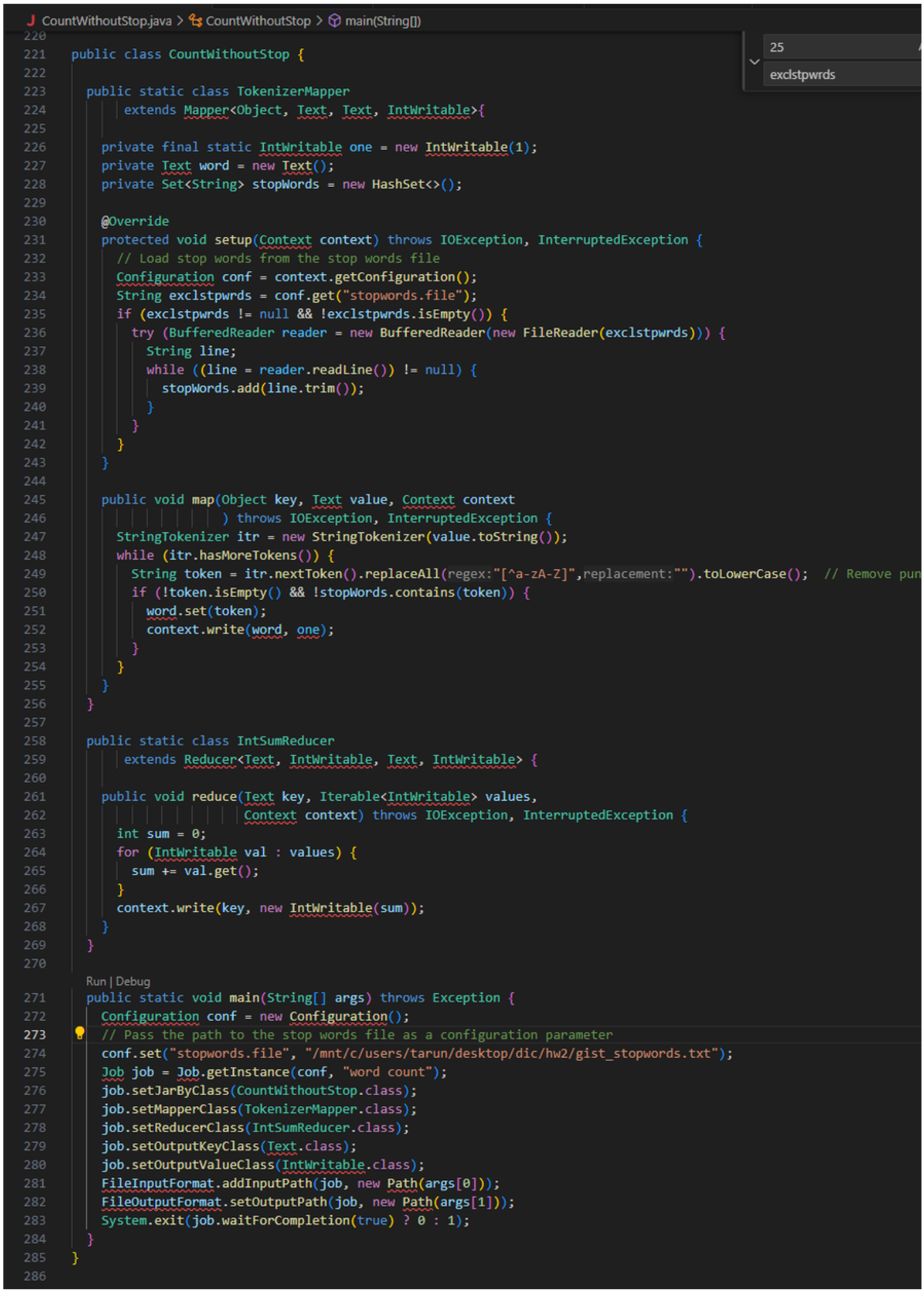
He're we need to modify the previous code to ensure we dont count the stop words, for doing this first need to load the stop words, i have taken stop words text file from internet, and in the code i created a method to load the stop words from the file, this adds the stop words from a file by trimming the line spaces between the words.

```
@Override
protected void setup(Context context) throws IOException, InterruptedException {
  // Load stop words from the stop words file
  Configuration conf = context.getConfiguration();
  String exclstpwrds = conf.get("stopwords.file");
  if (exclstpwrds != null && !exclstpwrds.isEmpty()) {
    try (BufferedReader reader = new BufferedReader(new FileReader(exclstpwrds))) {
      String line;
      while ((line = reader.readLine()) != null) {
        stopWords.add(line.trim());
      }
    }
  }
}
```

and in the main function define the path to the stop words file

```
Run | Debug
public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  // Pass the path to the stop words file as a configuration parameter
  conf.set("stopwords.file", "/mnt/c/users/tarun/desktop/dic/hw2/gist_stopwords.txt");
```

Now let’s look at the whole code


CountWithoutStop.java > CountWithoutStop > main(String[])
25
exclstpwrds
public class CountWithoutStop {
public static class TokenizerMapper
extends Mapper<Object, Text, Text, IntWritable>{
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
private Set<String> stopWords = new HashSet<>();
@Override
protected void setup(Context context) throws IOException, InterruptedException {
// Load stop words from the stop words file
Configuration conf = context.getConfiguration();
String exclstpwrds = conf.get("stopwords.file");
if (exclstpwrds != null && !exclstpwrds.isEmpty()) {
try (BufferedReader reader = new BufferedReader(new FileReader(exclstpwrds))) {
String line;
while ((line = reader.readLine()) != null) {
stopWords.add(line.trim());
}
}
}
}
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
StringTokenizer itr = new StringTokenizer(value.toString());
while (itr.hasMoreTokens()) {
String token = itr.nextToken().replaceAll(regex:"[^a-zA-Z]",replacement:"").toLowerCase(); // Remove pun
if (!token.isEmpty() && !stopWords.contains(token)) {
word.set(token);
context.write(word, one);
}
}
}
}
public static class IntSumReducer
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));
}
}
Run | Debug
public static void main(String[] args) throws Exception {
Configuration conf = new Configuration();
// Pass the path to the stop words file as a configuration parameter
conf.set("stopwords.file", "/mnt/c/users/tarun/desktop/dic/hw2/gist_stopwords.txt");
Job job = Job.getInstance(conf, "word count");
job.setJarByClass(CountWithoutStop.class);
job.setMapperClass(TokenizerMapper.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);
}
}

Glimpse of stop words file

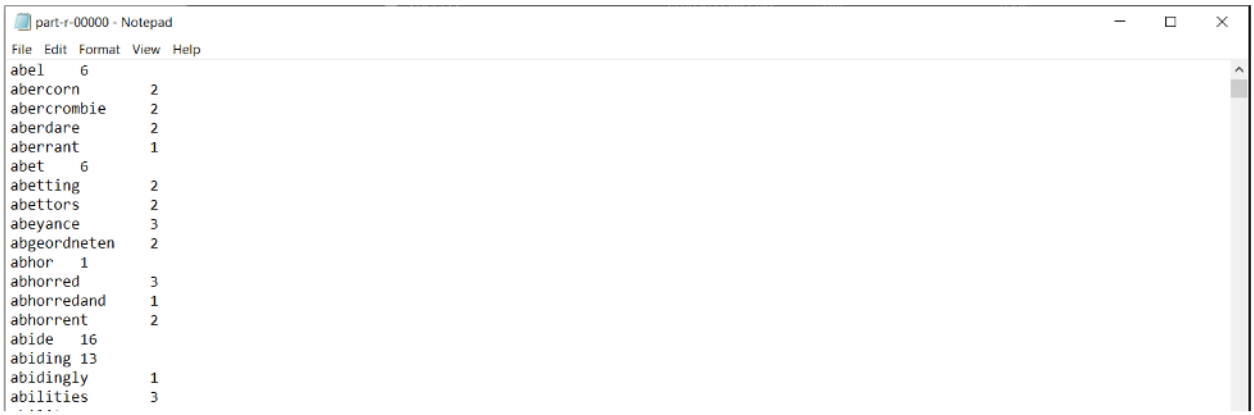
Initial Examination

Now let's run the program. First Compile the Java code for MapReduce: `javac -classpath 'hadoop classpath' -d /mnt/c/users/tarun/desktop/dic/hw2/count_w_stp/ CountWithoutStop.java`

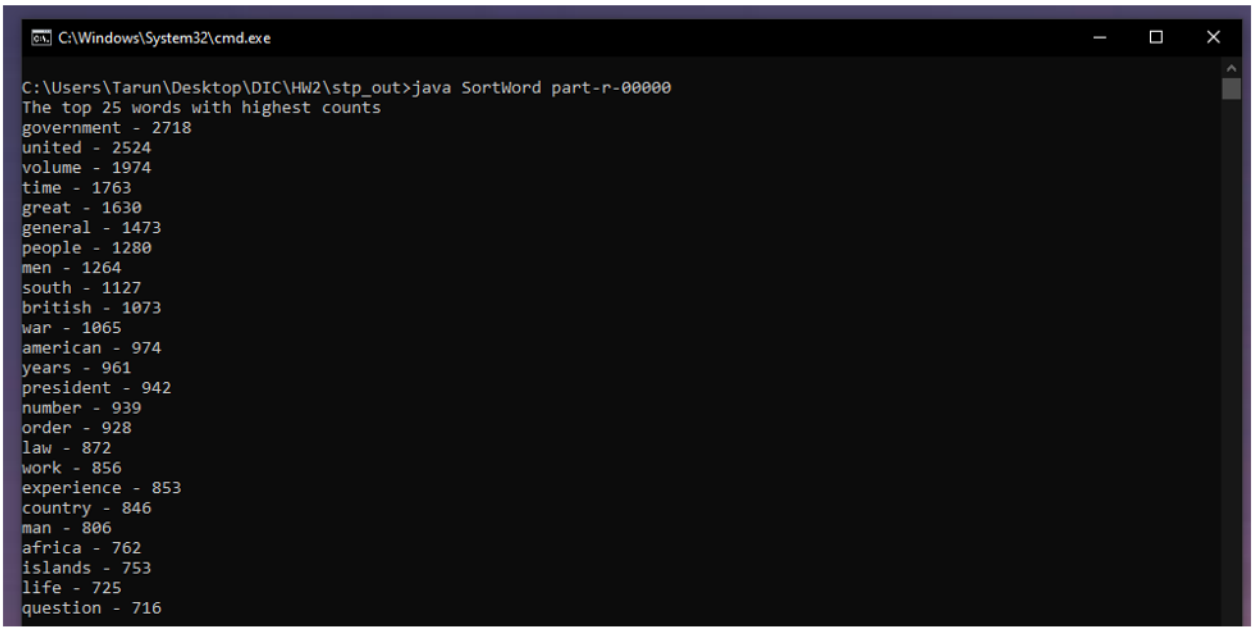
Then create a JAR File so .class files will be compiled files into a JAR `jar cf cwst.jar CountWithoutStop*.class` **Finally execute the jar file by running** `hadoop jar 'cwst.jar' CountWithoutStop /mydata /output_cwst`

```
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2$ cd count_w_stp/
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp$ jar cf cwst.jar CountWithoutStop*.class
kiyo@MININT-7B4IC2U:/mnt/c/users/tarun/desktop/dic/hw2/count_w_stp$ hadoop jar  'cwst.jar' CountWithoutStop /mydata /output_cwst
2023-11-04 20:19:23,157 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java cl
asses where applicable
2023-11-04 20:19:23,658 INFO client.RMProxy: Connecting to ResourceManager at localhost/127.0.0.1:8032
2023-11-04 20:19:24,062 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool in
terface and execute your application with ToolRunner to remedy this.
2023-11-04 20:19:24,091 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/kiyo/.stagi
ng/job_1698962573900_0019
2023-11-04 20:19:24,209 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
2023-11-04 20:19:24,376 INFO input.FileInputFormat: Total input files to process : 5
2023-11-04 20:19:24,425 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
2023-11-04 20:19:24,459 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
2023-11-04 20:19:24,484 INFO mapreduce.JobSubmitter: number of splits:5
2023-11-04 20:19:24,568 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted
= false
2023-11-04 20:19:24,584 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1698962573900_0019
2023-11-04 20:19:24,585 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-11-04 20:19:24,814 INFO conf.Configuration: resource-types.xml not found
2023-11-04 20:19:24,816 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-11-04 20:19:24,927 INFO impl.YarnClientImpl: Submitted application application_1698962573900_0019
2023-11-04 20:19:24,975 INFO mapreduce.Job: The url to track the job: http://MININT-7B4IC2U.localdomain:8088/proxy/application_1698
```

Now let's get the output files form the Hadoop to local `hadoop fs -copyToLocal /output_cwst/*`



As these are sorted by the key, let's sort this based on their count by using simple sort program in java, shown in previous section.

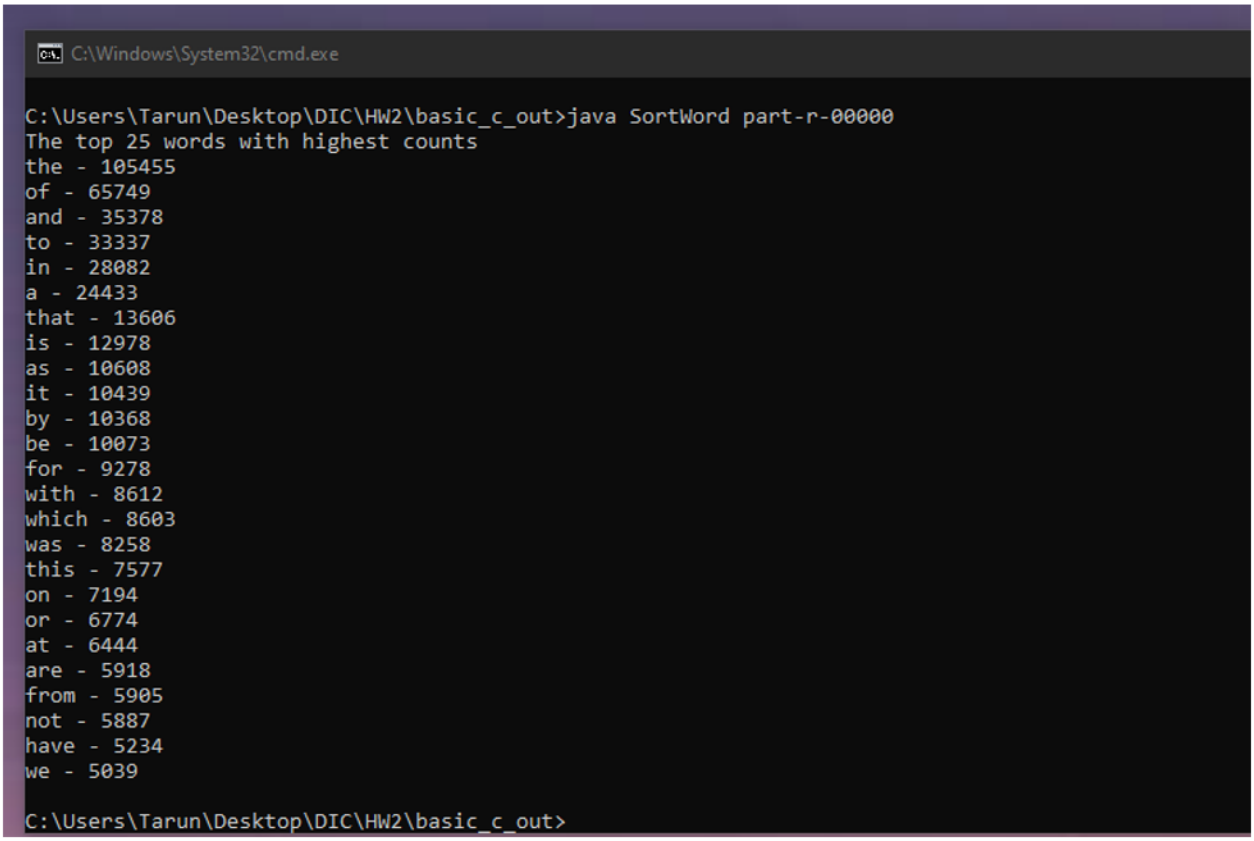


Analysis [Questions 1 - 6]

1. **What are the 25 most common words and the number of occurrences of each when you**
   do not remove stopwords?

Answer:

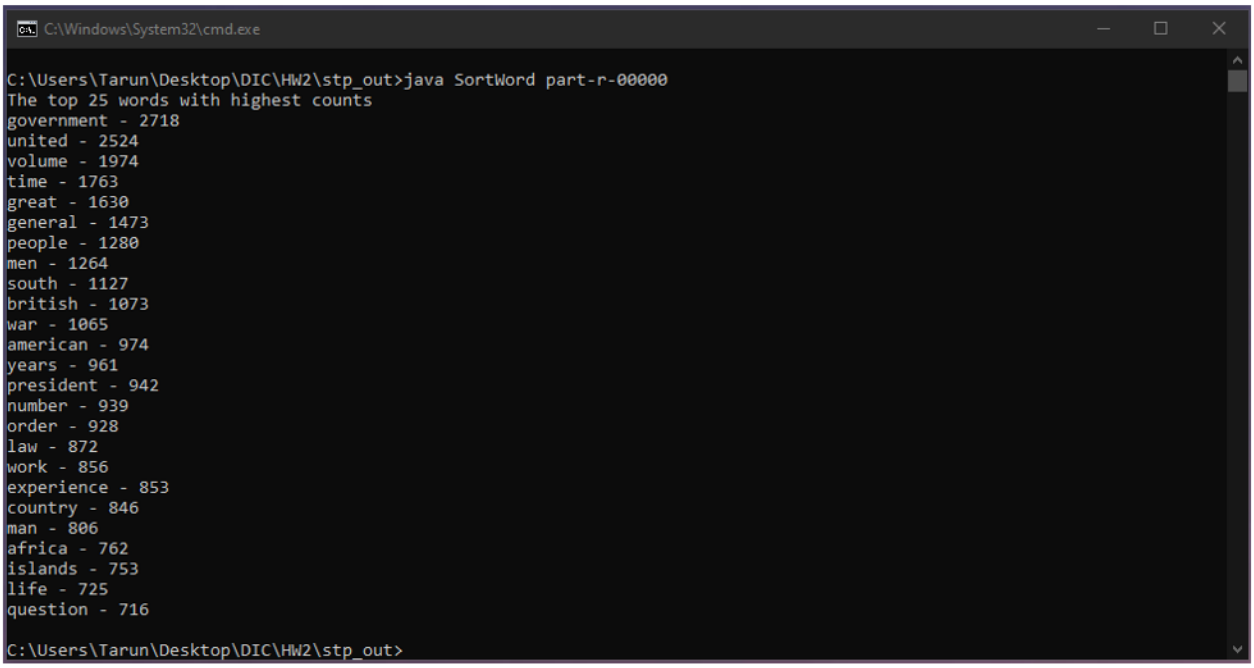
Here are 25 most common words and the number of occurrences of each when you do not remove stopwords:



2. **What are the 25 most common words and the number of occurrences of each when you**
   do remove stopwords?

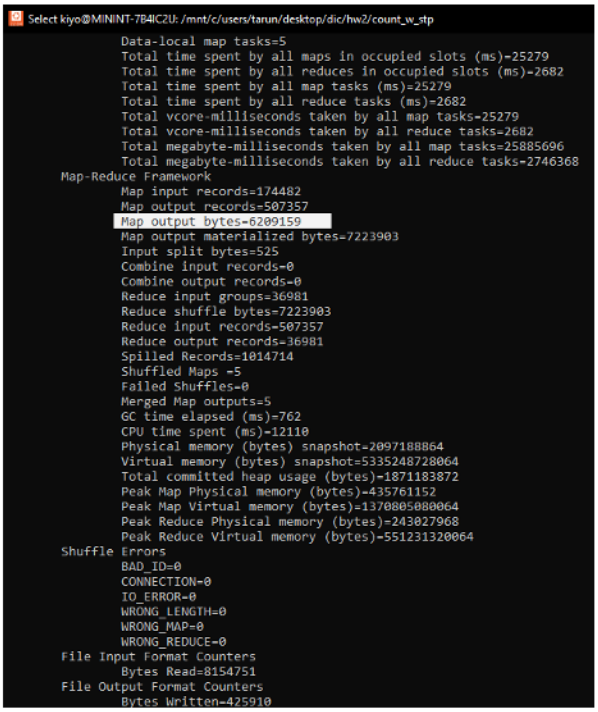
Answer:

Here are 25 most common words and the number of occurrences of each when we remove stopwords:

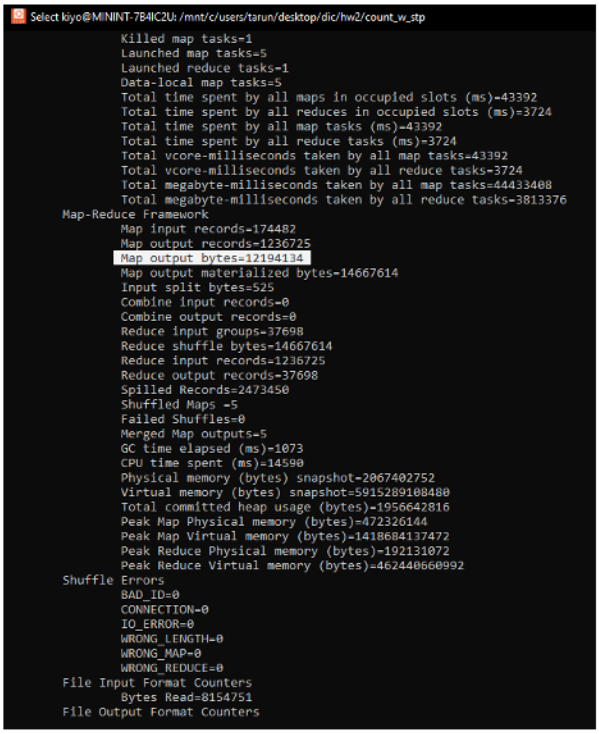


3. **Based on the output of your application, how does removing stop words affect the total**
   amount of bytes output by your mappers? Name one concrete way that this would affect
   the performance of your application.

- Without stop words:
  - Total map output bytes: 6,209,159 bytes
  - File Output Bytes Written=433133 bytes
- Data with stop words:
  - Total map output bytes: 12,194,134 bytes
  - File Output Bytes Written=425910



Without Stop Words



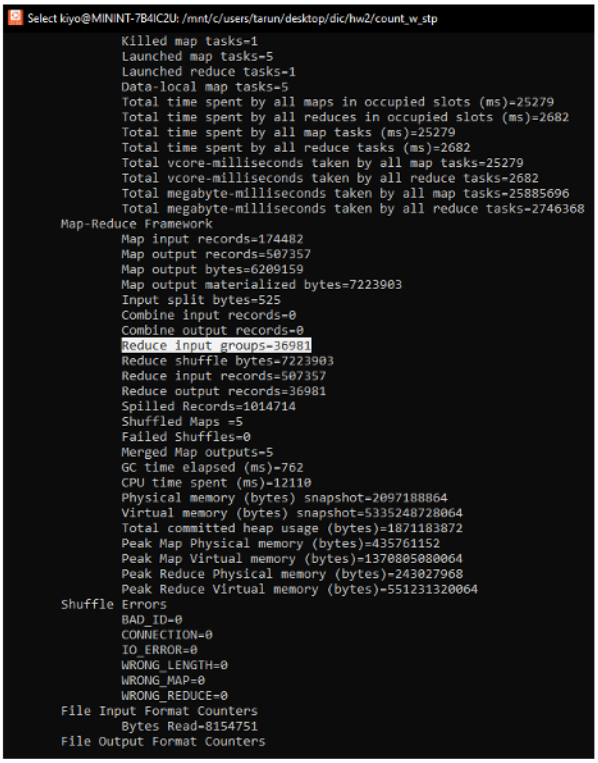
With Stop Words

The removal of stopwords can bring significant performance increase it reduces the data that mappers need to transmit to the reducers, as we can see the mappers have 6,209,159 bytes to send, compared to 12,194,134 bytes with stopwords. This can lead to faster data processing and increased scalability as the size of the data decreases without stopwords as you can see the File Output Bytes Written=425910 with stopwords is higher than the File Output Bytes Written=433133 bytes without stopwords. Finally, as there's less data to process there's less usage of the resources like memory. Hence removing the stop words helps in overall increase increase in the performance of the system.

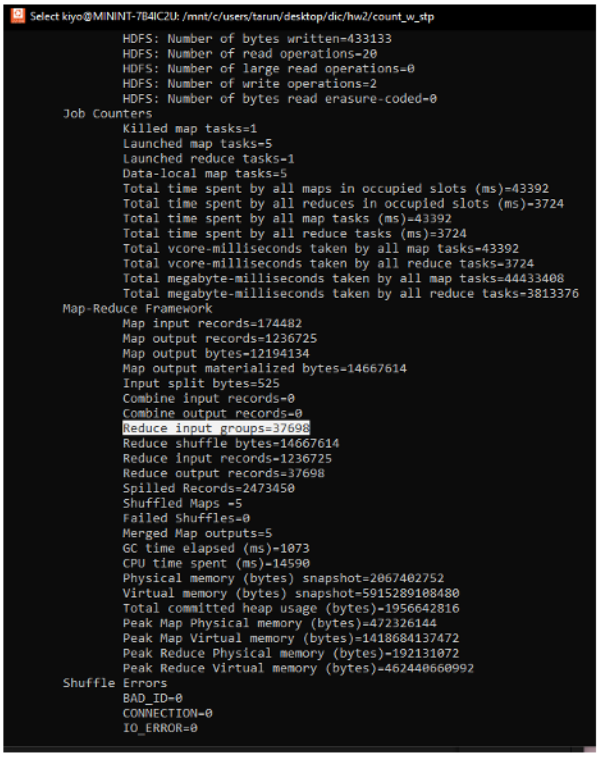
4. **Based on the output of your application, what is the size of your keyspace with and**
   without removing stopwords? How does this correspond to the number of stopwords you
   have chosen to remove?

- Number of stopwords used in gist_stopwords.txt: 851

- Data without stop words:
  - Total reduce input groups (keyspace size): 36,981
- Data with stop words:
  - Total reduce input groups (keyspace size): 37,698



Without STOP



With STOP

Without stop words, the keyspace is 36,981, while with stop words, it is 37,698. The keyspace size is proportional to the number of unique words in the dataset.

So total number of stop words reduced: 37,698 - 36,981 = 717.

5. **Let's now assume you were going to run your application on the entirety of Project Gutenberg. For this question, assume that there are 100TB of input data, the data is spread over 10 sites, and each site has 20 mappers. Assume you ignore all but the 25**

most common words that you listed in question 2. Furthermore, assume that your
combiners have been run optimally so that each combiner will output at most 1 keyvalue pair per key.

- **How much data will each mapper have to parse?**
- **What is the size of your keyspace?**
- **What is the maximum number of key-value pairs that could be communicated**
  during the barrier between the mapping and reducing?
- **Assume you are running one reducer per site. On average, how many key-value**
  pairs will each reducer have to handle?

Answer:

- **How much data will each mapper have to parse?**

  Given,

  - 100 TB of input data
  - Number of sites the input data is being spread = 10
  - Number of mappers = 20

  So, Let's calculate the data per site = Total input data / sites

  - 100TB / 10 = 10 TB of data per site

  Data that each mapper needs to parse = Data given per site / the number of mappers

  - 10TB / 20 = 500GB

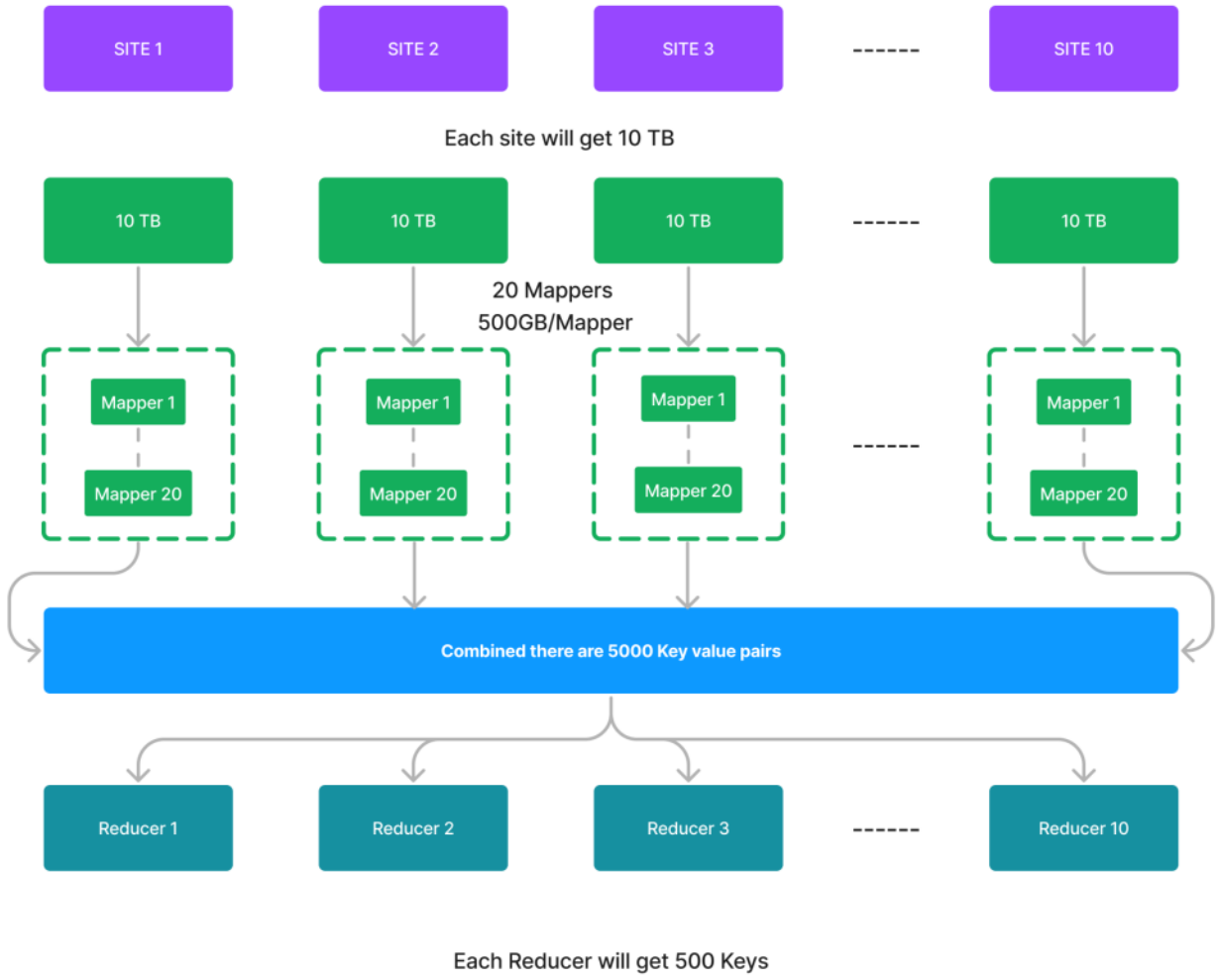
  SO each mapper will have 500GB or 500000MB of data to parse.

- **What is the size of your keyspace?**
  - 25 as all the words are being excluded.
- **What is the maximum number of key-value pairs that could be communicated during the barrier between the mapping and reducing?**

- There are 25 keys per mapper
- There are 20 Mappers
- There are 10 sites
- So As each mapper may output at most 1 key pair
- total key value pairs = 25 keys per mapper x 20 mappers per site x 10 sites = 5000 key value pairs.

- **Assume you are running one reducer per site. On average, how many key-value**
  pairs will each reducer have to handle?
  - Since there are 5000 key value pairs for and 10 sites, so each reducer will handle 500 key value pairs worth of data.

6. **Draw the data flow diagram for question 5. The diagram should be similar to the diagram**
   shown in the lecture. On your diagram, label the specific quantities you got for 5a,b,c,
   and d.



References

- *Easiest way to sort a list of words by occurance*. (n.d.). Stack Overflow. Retrieved November 6, 2023, from https://stackoverflow.com/questions/2366443/easiest-way-to-sort-a-list-of-words-by-occurance

- *Enhancing the word count sample a little bit*. (2012, October 28). Altocumulus. https://yossidahan.wordpress.com/2012/10/28/enhancing-the-word-count-sample-a-little-bit/
- *How to copy file from HDFS to the local file system*. (n.d.). Stack Overflow. Retrieved November 6, 2023, from https://stackoverflow.com/questions/17837871/how-to-copy-file-from-hdfs-to-the-local-file-system
- *MapReduce Tutorial*. (n.d.). Apache.org. Retrieved November 6, 2023, from https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html
- *[Solved]-Remove punctuation and HTML entity with hadoop Wordcount in java-Java*. (n.d.). Hire Developers, Free Coding Resources for the Developer. Retrieved November 6, 2023, from https://www.appsloveworld.com/java/100/2159/remove-punctuation-and-html-entity-with-hadoop-wordcount-in-java
- *Stop words list*. (n.d.). CountWordsFree. Retrieved November 6, 2023, from https://countwordsfree.com/stopwords