1. // Libraries - Additional are the Job control libraries

**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.

**Output**: 100 Most frequent words

2. **import** org.apache.hadoop.conf.Configuration;
3. **import** org.apache.hadoop.conf.Configured;
4. **import** org.apache.hadoop.fs.Path;
5. **import** org.apache.hadoop.io.IntWritable;
6. **import** org.apache.hadoop.io.Text;
7. **import** org.apache.hadoop.mapreduce.Job;
8. **import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9. **import** org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
10. **import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
11. **import** org.apache.hadoop.util.Tool;
12. **import** org.apache.hadoop.util.ToolRunner;
13. **import** org.apache.hadoop.mapreduce.lib.jobcontrol.ControlledJob;
14. **import** org.apache.hadoop.mapreduce.lib.jobcontrol.JobControl;
15. // First Mapper Class
17. **public** **class** WordMapper **extends** Mapper<LongWritable, Text, Text, IntWritable> {
19. IntWritable intWritable = **new** IntWritable(1);
20. Text text = **new** Text();
22. @Override
23. **public** **void** map(LongWritable key, Text value, Context context) **throws** IOException, InterruptedException {
25. // Filtering the text without numbers or spcial characters
27. String line = value.toString().replaceAll("[^\\p{L}\\p{Z}]","");
29. **for** (String word : line.split("\\W+")) {
30. **if** (word.length() > 0) {
31. text.set(word);
32. context.write(text, intWritable);
33. }
34. }
35. }
36. }
38. // First Reducer Class
40. **public** **class** SumReducer **extends** Reducer<Text, IntWritable, Text, IntWritable> {
42. IntWritable intWritable = **new** IntWritable();
44. @Override
45. **public** **void** reduce(Text key, Iterable<IntWritable> values, Context context) **throws** IOException, InterruptedException {
47. **int** wordCount = 0;
48. **for** (IntWritable value : values) {
49. wordCount += value.get();
50. }
51. intWritable.set(wordCount);
52. context.write(key, intWritable);
53. }
54. }
56. // Second Mapper Class -- Swaps the output of first reducer
58. **public** **class** WordMapper2 **extends** Mapper< Text, Text, IntWritable, Text> {
60. IntWritable frequency = **new** IntWritable();
62. @Override
63. **public** **void** map(Text key, Text value, Context context)
64. **throws** IOException, InterruptedException {
66. **int** newVal = Integer.parseInt(value.toString());
67. frequency.set(newVal);
68. context.write(frequency, key);
69. }
70. }
72. // Second reducer CLass
74. **public** **class** SumReducer2 **extends** Reducer<IntWritable, Text, IntWritable, Text> {
76. Text word = **new** Text();
78. @Override
79. **public** **void** reduce(IntWritable key, Iterable<Text> values, Context context)
80. **throws** IOException, InterruptedException {
82. **for** (Text value : values) {
83. word.set(value);
84. context.write(key, word);
85. }
86. }
87. }
88. // Comparator Class – For Descending Order.
89. **package** stubs;
90. **import** java.nio.ByteBuffer;
91. **import** org.apache.hadoop.io.IntWritable;
92. **import** org.apache.hadoop.io.WritableComparator;
94. **public** **class** IntComparator **extends** WritableComparator {
96. **public** IntComparator() {
97. **super**(IntWritable.**class**);
98. }
100. @Override
101. **public** **int** compare(**byte**[] b1, **int** s1, **int** l1, **byte**[] b2,
102. **int** s2, **int** l2) {
103. Integer v1 = ByteBuffer.wrap(b1, s1, l1).getInt();
104. Integer v2 = ByteBuffer.wrap(b2, s2, l2).getInt();
105. **return** v1.compareTo(v2) \* (-1);
106. }
107. }
108. // Driver Class
110. **public** **class** WordCombined **extends** Configured **implements** Tool {
112. **public** **int** run(String[] args) **throws** Exception {
114. JobControl jobControl = **new** JobControl("jobChain");
115. Configuration conf1 = getConf();
117. Job job1 = Job.getInstance(conf1);
118. job1.setJarByClass(WordCombined.**class**);
119. job1.setJobName("Word Combined");
121. FileInputFormat.setInputPaths(job1, **new** Path(args[0]));
122. FileOutputFormat.setOutputPath(job1, **new** Path(args[1] + "/temp"));
124. //Job 1
125. job1.setMapperClass(WordMapper.**class**);
126. job1.setReducerClass(SumReducer.**class**);
127. job1.setCombinerClass(SumReducer.**class**);
129. job1.setOutputKeyClass(Text.**class**);
130. job1.setOutputValueClass(IntWritable.**class**);
132. ControlledJob controlledJob1 = **new** ControlledJob(conf1);
133. controlledJob1.setJob(job1);
135. jobControl.addJob(controlledJob1);
136. Configuration conf2 = getConf();
138. //Job2
140. Job job2 = Job.getInstance(conf2);
141. job2.setJarByClass(WordCombined.**class**);
142. job2.setJobName("Word Invert");
144. FileInputFormat.setInputPaths(job2, **new** Path(args[1] + "/temp"));
145. FileOutputFormat.setOutputPath(job2, **new** Path(args[1] + "/final"));
147. job2.setMapperClass(WordMapper2.**class**);
148. job2.setReducerClass(SumReducer2.**class**);
149. job2.setCombinerClass(SumReducer2.**class**);
151. job2.setOutputKeyClass(IntWritable.**class**);
152. job2.setOutputValueClass(Text.**class**);
153. job2.setInputFormatClass(KeyValueTextInputFormat.**class**);
155. job2.setSortComparatorClass(IntComparator.**class**);
156. ControlledJob controlledJob2 = **new** ControlledJob(conf2);
157. controlledJob2.setJob(job2);
159. // make job2 dependent on job1
160. controlledJob2.addDependingJob(controlledJob1);
162. // add the job to the job control
163. jobControl.addJob(controlledJob2);
164. Thread jobControlThread = **new** Thread(jobControl);
165. jobControlThread.start();
167. // Printing the Job States
169. **while** (!jobControl.allFinished()) {
170. System.out.println("Jobs in waiting state: " + jobControl.getWaitingJobList) .size());
171. System.out.println("Jobs in ready state: " + jobControl.getReadyJobsList().size());
172. System.out.println("Jobs in running state: " + jobControl.getRunningJobList().size());
173. System.out.println("Jobs in success state: " + jobControl.getSuccessfulJobList().size());
174. System.out.println("Jobs in failed state: " + jobControl.getFailedJobList().size());
175. **try** {
176. Thread.sleep(5000);
177. } **catch** (Exception e) {
179. }
181. }
182. System.exit(0);
183. **return** (job1.waitForCompletion(**true**) ? 0 : 1);
184. }
185. **public** **static** **void** main(String[] args) **throws** Exception {
186. **int** exitCode = ToolRunner.run(**new** WordCombined(), args);
187. System.exit(exitCode);
188. }
189. }