**LABORATORY PROGRAM – 7**

**Create a Map Reduce program to sort the content in**

**an alphabetic order listing only top 10 maximum occurrences of words**

**Driver Code (TopNDriver.java)**

package samples.topn;

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.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TopNDriver {

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

if (args.length != 3) {

System.err.println("Usage: TopNDriver <in> <temp-out> <final-out>");

System.exit(2);

}

Configuration conf = new Configuration();

// === Job 1: Word Count ===

Job wcJob = Job.getInstance(conf, "word count");

wcJob.setJarByClass(TopNDriver.class);

wcJob.setMapperClass(WordCountMapper.class);

wcJob.setCombinerClass(WordCountReducer.class);

wcJob.setReducerClass(WordCountReducer.class);

wcJob.setOutputKeyClass(Text.class);

wcJob.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(wcJob, new Path(args[0]));

Path tempDir = new Path(args[1]);

FileOutputFormat.setOutputPath(wcJob, tempDir);

if (!wcJob.waitForCompletion(true)) {

System.exit(1);

}

// === Job 2: Top N ===

Job topJob = Job.getInstance(conf, "top 10 words");

topJob.setJarByClass(TopNDriver.class);

topJob.setMapperClass(TopNMapper.class);

topJob.setReducerClass(TopNReducer.class);

topJob.setMapOutputKeyClass(IntWritable.class);

topJob.setMapOutputValueClass(Text.class);

topJob.setOutputKeyClass(Text.class);

topJob.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(topJob, tempDir);

FileOutputFormat.setOutputPath(topJob, new Path(args[2]));

System.exit(topJob.waitForCompletion(true) ? 0 : 1);

}

}

**Mapper Code (WordCountMapper.java)**

package samples.topn;

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class WordCountMapper

extends Mapper<Object, Text, Text, IntWritable> {

private final static IntWritable ONE = new IntWritable(1);

private Text word = new Text();

// characters to normalize into spaces

private String tokens = "[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']";

@Override

protected void map(Object key, Text value, Context context)

throws IOException, InterruptedException {

// clean & tokenize

String clean = value.toString()

.toLowerCase()

.replaceAll(tokens, " ");

StringTokenizer itr = new StringTokenizer(clean);

while (itr.hasMoreTokens()) {

word.set(itr.nextToken().trim());

context.write(word, ONE);

}

}

}

**Mapper Code (TopNMapper.java)**

package samples.topn;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TopNMapper

extends Mapper<Object, Text, IntWritable, Text> {

private IntWritable count = new IntWritable();

private Text word = new Text();

@Override

protected void map(Object key, Text value, Context context)

throws IOException, InterruptedException {

// input line: word \t count

String[] parts = value.toString().split("\\t");

if (parts.length == 2) {

word.set(parts[0]);

count.set(Integer.parseInt(parts[1]));

// emit count → word, so Hadoop sorts by count

context.write(count, word);

}

}

}

**Reducer Code (WordCountReducer.java)**

package samples.topn;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class WordCountReducer

extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

protected 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));

}

}

**Reducer Code (TopNReducer.java)**

package samples.topn;

import java.io.IOException;

import java.util.ArrayList;

import java.util.Collections;

import java.util.List;

import java.util.Map;

import java.util.TreeMap;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class TopNReducer

extends Reducer<IntWritable, Text, Text, IntWritable> {

// TreeMap with descending order of keys (counts)

private TreeMap<Integer, List<String>> countMap =

new TreeMap<>(Collections.reverseOrder());

@Override

protected void reduce(IntWritable key, Iterable<Text> values, Context context)

throws IOException, InterruptedException {

int cnt = key.get();

List<String> words = countMap.getOrDefault(cnt, new ArrayList<>());

for (Text w : values) {

words.add(w.toString());

}

countMap.put(cnt, words);

}

@Override

protected void cleanup(Context context)

throws IOException, InterruptedException {

// collect top 10 word→count pairs

List<WordCount> topList = new ArrayList<>();

int seen = 0;

for (Map.Entry<Integer, List<String>> entry : countMap.entrySet()) {

int cnt = entry.getKey();

for (String w : entry.getValue()) {

topList.add(new WordCount(w, cnt));

seen++;

if (seen == 10) break;

}

if (seen == 10) break;

}

// sort these 10 entries alphabetically by word

Collections.sort(topList, (a, b) -> a.word.compareTo(b.word));

// emit final top 10 in alphabetical order

for (WordCount wc : topList) {

context.write(new Text(wc.word), new IntWritable(wc.count));

}

}

// helper class

private static class WordCount {

String word;

int count;

WordCount(String w, int c) { word = w; count = c; }

}

}

**OBSERVATION**





