**Java jar file direct :**

# Recompile (assuming your .java files are present)

javac -classpath $(hadoop classpath) -d . TDriver.java TMapper.java TReducer.java

# Create the JAR

jar -cvf temp.jar \*.class

mv temp.jar /home/hadoop/

hadoop jar /home/hadoop/temp.jar TDriver input output

**For Running Java Hadoop :**

hadoop fs -mkdir -p /user/hadoop/input

hadoop fs -copyFromLocal /home/hadoop/Desktop/sample.txt /user/hadoop/input/

hadoop fs -ls /user/hadoop/input

hadoop jar /home/hadoop/Desktop/temp.jar TDriver /user/hadoop/input /user/hadoop/output

hadoop fs -ls /user/hadoop/output

hadoop fs -cat /user/hadoop/output/part-r-00000

**if output exists remove :**

hadoop fs -rm -r /user/hadoop/output

**Year Wise Temperature :**

TMapper:

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

private Text year = new Text();

private IntWritable temperature = new IntWritable();

@Override

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

String line = value.toString();

// Check if line length is sufficient for parsing (e.g., NOAA weather data line length)

if (line.length() >= 93) {

// Extract year from chars 15 to 19 (index 15 to 18 inclusive)

String yearStr = line.substring(15, 19);

// Extract air temperature at positions 87 to 92 (indexes 87 to 91 inclusive)

String tempStr = line.substring(87, 92).trim();

// Temperature sign check: if char at 87 is '+', parse substring 88-92, else 87-92

// But here we assume the tempStr includes sign.

try {

int temp = Integer.parseInt(tempStr);

year.set(yearStr);

temperature.set(temp);

// Emit (year, temperature)

context.write(year, temperature);

} catch (NumberFormatException e) {

// Ignore invalid temperature

}

}

// else ignore lines that are too short

}

}

**TReducer :**

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

private IntWritable result = new IntWritable();

@Override

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

int maxTemp = Integer.MIN\_VALUE;

for (IntWritable val : values) {

if (val.get() > maxTemp) {

maxTemp = val.get();

}

}

result.set(maxTemp);

context.write(key, result);

}

}

**TDriver :**

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 TDriver {

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

if (args.length != 2) {

System.err.println("Usage: TDriver <input path> <output path>");

System.exit(-1);

}

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Temperature Max per Year");

job.setJarByClass(TDriver.class);

job.setMapperClass(TMapper.class);

job.setReducerClass(TReducer.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);

}

}

**MeanMax Temperature :**

**MMapper:**

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

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

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

String line = value.toString();

if (line.length() >= 21) {

try {

// Extract temperature and parse to int

String tempString = line.substring(19, 21);

int temp = Integer.parseInt(tempString);

context.write(new Text("Average Temperature"), new IntWritable(temp));

} catch (NumberFormatException e) {

// Skip lines with invalid temperature format

System.err.println("Invalid number format in line: " + line);

}

} else {

// Skip or log short lines

System.err.println("Skipping short line: " + line);

}

}

}

**MReducer :**

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

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

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

throws IOException, InterruptedException {

int sum = 0;

int count = 0;

for (IntWritable val : values) {

sum += val.get();

count++;

}

int average = (count == 0) ? 0 : sum / count;

context.write(key, new IntWritable(average));

}

}

**MDriver :**

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 MDriver {

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

Configuration conf = new Configuration();

Job job = Job.getInstance(conf, "Average Temperature");

job.setJarByClass(MDriver.class);

job.setMapperClass(MMapper.class);

job.setReducerClass(MReducer.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);

}

}

Top N and MeanMax :

Mam Notes :

1. TopN

**Driver-TopN.class**

package samples.topn;  
  
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.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.util.GenericOptionsParser;  
  
public class TopN {  
 public static void main(String[] args) throws Exception {  
 Configuration conf = new Configuration();  
 String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingArgs();  
 if (otherArgs.length != 2) {  
 System.err.println("Usage: TopN <in> <out>");  
 System.exit(2);  
 }   
 Job job = Job.getInstance(conf);  
 job.setJobName("Top N");  
 job.setJarByClass(TopN.class);  
 job.setMapperClass(TopNMapper.class);  
 job.setReducerClass(TopNReducer.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 FileInputFormat.addInputPath(job, new Path(otherArgs[0]));  
 FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }  
   
 public static class TopNMapper extends Mapper<Object, Text, Text, IntWritable> {  
 private static final IntWritable one = new IntWritable(1);  
   
 private Text word = new Text();  
   
 private String tokens = "[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']";  
   
 public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 String cleanLine = value.toString().toLowerCase().replaceAll(this.tokens, " ");  
 StringTokenizer itr = new StringTokenizer(cleanLine);  
 while (itr.hasMoreTokens()) {  
 this.word.set(itr.nextToken().trim());  
 context.write(this.word, one);  
 }   
 }  
 }  
}

TopNCombiner.class

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

**TopNMapper.class**

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 TopNMapper extends Mapper<Object, Text, Text, IntWritable> {  
 private static final IntWritable one = new IntWritable(1);  
   
 private Text word = new Text();  
   
 private String tokens = "[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']";  
   
 public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 String cleanLine = value.toString().toLowerCase().replaceAll(this.tokens, " ");  
 StringTokenizer itr = new StringTokenizer(cleanLine);  
 while (itr.hasMoreTokens()) {  
 this.word.set(itr.nextToken().trim());  
 context.write(this.word, one);  
 }   
 }  
}

**TopNReducer.class**

package samples.topn;  
  
import java.io.IOException;  
import java.util.HashMap;  
import java.util.Map;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
import utils.MiscUtils;  
  
public class TopNReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
 private Map<Text, IntWritable> countMap = new HashMap<>();  
   
 public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 int sum = 0;  
 for (IntWritable val : values)  
 sum += val.get();   
 this.countMap.put(new Text(key), new IntWritable(sum));  
 }  
   
 protected void cleanup(Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 Map<Text, IntWritable> sortedMap = MiscUtils.sortByValues(this.countMap);  
 int counter = 0;  
 for (Text key : sortedMap.keySet()) {  
 if (counter++ == 20)  
 break;   
 context.write(key, sortedMap.get(key));  
 }   
 }  
}  
  
**2.MeanMax**

**MeanMaxDriver.class**

package meanmax;  
  
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 MeanMaxDriver {  
 public static void main(String[] args) throws Exception {  
 if (args.length != 2) {  
 System.err.println("Please Enter the input and output parameters");  
 System.exit(-1);  
 }   
 Job job = new Job();  
 job.setJarByClass(MeanMaxDriver.class);  
 job.setJobName("Max temperature");  
 FileInputFormat.addInputPath(job, new Path(args[0]));  
 FileOutputFormat.setOutputPath(job, new Path(args[1]));  
 job.setMapperClass(MeanMaxMapper.class);  
 job.setReducerClass(MeanMaxReducer.class);  
 job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }  
}  
  
**MeanMaxMapper.class**

package meanmax;  
  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Mapper;  
  
public class MeanMaxMapper extends Mapper<LongWritable, Text, Text, IntWritable> {  
 public static final int MISSING = 9999;  
   
 public void map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 int temperature;  
 String line = value.toString();  
 String month = line.substring(19, 21);  
 if (line.charAt(87) == '+') {  
 temperature = Integer.parseInt(line.substring(88, 92));  
 } else {  
 temperature = Integer.parseInt(line.substring(87, 92));  
 }   
 String quality = line.substring(92, 93);  
 if (temperature != 9999 && quality.matches("[01459]"))  
 context.write(new Text(month), new IntWritable(temperature));   
 }  
}  
  
**MeanMaxReducer.class**

package meanmax;  
  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
  
public class MeanMaxReducer extends Reducer<Text, IntWritable, Text, IntWritable> {  
 public void reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) throws IOException, InterruptedException {  
 int max\_temp = 0;  
 int total\_temp = 0;  
 int count = 0;  
 int days = 0;  
 for (IntWritable value : values) {  
 int temp = value.get();  
 if (temp > max\_temp)  
 max\_temp = temp;   
 count++;  
 if (count == 3) {  
 total\_temp += max\_temp;  
 max\_temp = 0;  
 count = 0;  
 days++;  
 }   
 }   
 context.write(key, new IntWritable(total\_temp / days));  
 }  
}

**My Codes :**

**TopNDriver :**

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;

import org.apache.hadoop.util.GenericOptionsParser;

public class TopN {

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

Configuration conf = new Configuration();

String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();

if (otherArgs.length < 2) {

System.err.println("Usage: TopN <in> <out> [N]");

System.exit(2);

}

// Default top N value

int N = 20;

if (otherArgs.length == 3) {

N = Integer.parseInt(otherArgs[2]);

}

conf.setInt("top.n", N);

Job job = Job.getInstance(conf, "Top N");

job.setJarByClass(TopN.class);

job.setMapperClass(TopNMapper.class);

job.setReducerClass(TopNReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(otherArgs[0]));

FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

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

}

}

**TopNMapper :**

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

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

private Text word = new Text();

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

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

String cleanLine = value.toString().toLowerCase().replaceAll(tokens, " ");

StringTokenizer itr = new StringTokenizer(cleanLine);

while (itr.hasMoreTokens()) {

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

context.write(word, one);

}

}

}

**TopNReducer :**

package samples.topn;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import java.util.Map.Entry;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import utils.MiscUtils;

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

private Map<Text, IntWritable> countMap = new HashMap<>();

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

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

countMap.put(new Text(key), new IntWritable(sum));

}

protected void cleanup(Context context) throws IOException, InterruptedException {

int N = context.getConfiguration().getInt("top.n", 20);

Map<Text, IntWritable> sortedMap = MiscUtils.sortByValues(countMap);

int counter = 0;

for (Entry<Text, IntWritable> entry : sortedMap.entrySet()) {

if (counter++ == N)

break;

context.write(entry.getKey(), entry.getValue());

}

}

}