LAB-4

Find avg temperature for each year for NCDC dataset

```
package temp;
public class Average Mapper extends Mapper<long
    Writable, Text, Text, IntWritable> (context context)
throws IOException, Interrupted Exception {
 int temperature;
 String line = value.toString()
 String year = line.substring(15,19);
 if (line.charAt(8) == '+') {
 temperature = Integer.parseInt(line.substring(89,92));
}else{
     temperature = Integer.parseInt(line.substring(87,92));
}
 String quality = line.substring(92,93);
 if(temperature != 9999 && quality.matches("[01459]"))
   context.write(new Text(year), new IntWritable(temperature));
   }
}
public class Average Reduce extends Reducer<Text, IntWritable,
   Text, IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values,
   Reduce<Text, IntWritable, Text, IntWritable>.context context)
   throws IOException, Interrupted Exception {
          int max_temp = 0;
          int count = 0;
          for(IntWritable value : value) {
                max_temp += value.get()
                count++;
          }
     context.write(key, new IntWritable(max_temp/count));
   }
}
```

output
hdfs dfs -cat /avg temp - output dir / part - r - 00000
190 46

) Find the mean max temp for every month

```
package meanmax;
public class MeanMaxMapper extends Mapper<Long
  Writable, Text, Text, IntWritable> {
      public static final int MISSING = 9999;
public void map (LongWritable key, Text value, Mapper
  <Long Writable, Text, Text, IntWritable > Context context)
  throws IOException, InterruptedException {
      int temperature;
      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));
}}
public class MeanMaxReduce extends Reduce
  < 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 > 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));
}}
```

O/P -

```
hdfs dfs -cat /mean_max_output/*
```

01	4
02	0
03	7
05	100
06	198

LAB 5

Create a Mapreduce program to sort content in an alphabetical order listing only top 10 maximum occurrence of words

```
public class TopMapper 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, Interrupted Exception {
      String cleanLine = value.toString().toLower(
                    replaceAll (this.tokens, " "));
      ~~String Tokenize itr = new To next Toke().trim()~~
      String Tokenize itr = new StringTokenize (cleanLine);
      while (itr.hasMoreTokens()) {
        this.word.set(itr.nextToken().trim());
        context.write(this.word, one);

      } }
}
public class topNReduce extends Reducer<Text, Int
  Writable, text, IntWritable> {
  private Map < Text, IntWritable> countMap =
                              new HashMap<>();
  public void reduce(Text key, Iterable<IntWritable>
                              values;
    Reducer<Text, Int Writable, Text, IntWritable>
      context context)
    throws IO Exception, Interrupted Exception {
        int sum = 0;

    for (IntWritable val: values)
        sum += val.get();
```

```
        this.countMap.put(new Text(key), new IntWritable(sum));
    }
    protected void cleanup(Reducer.Context<Text, IntWritable, Text,
    IntWritable> context) throws IOException,
    InterruptedException {
        Map<Text, IntWritable> SortedMap = MiscUtils.SortByValue
        (this.countMap);
        int counter = 0;
        for(Text key: SortedMap.keySet()) {
            if (counter++ == 20)
                break;
            context.write(key, SortedMap.get(key));
        }
    }
}
```

Output

```
hdfs dfs -cat /output_dis/*
hello   2
hadoop  1
world   1
bye     1
```

Lab 6

Create MapReduce program to demonstrate join operations

Mapper

```
public void map(LongWritable key, Text value, OutputCollector
        <Textpair, Text> Output, Reporter)
        throws IOException
    {
        String v = Value.toString[];
        String[] sud = v.split("\t");
        Output.collect(new Textpair[sud[i]","i"]
                newText(sud[1]));
    }
```

Reducer :

```
public void reduce(Textpair, key, Iterable<Text> value,
        Output Collection<Text, Text> Output, Reporter rep)
        throws IOException
    {   Text nodeId = new Text(Values.next());
        which(Values.nu text[1])
        {   text node = Valuesnext();
            Text univalue = new text(nodeId.toString()
                    "||||"| node.toString());

            Output.collect(key getFirst(), output Value);

        }
    }
```

O/P $ hadoop jar throws /hd/joni.jar /input /data.txt /output

$ hadoop dfs -cat /output -part-r-00000

A_{11}	"2"	"36134"
B_{12}	"2"	"76"
C_{13}	"0"	"6354"

Lab 7

Print Word Count on Scala Shell & print Helloworld on Scala IDE

(a) Word Count on Scala ~~Scala~~ Shell

```
val data = sc.textFile("Spark.txt")
data.collect;
val Splitdata = data.flatMap(line => line.split(" "));
Split data.collect;
val mapdata = Split data.map(word => (word,1));
map data.collect;
val reduce data = mapdata.reduceByKey(_+_);
reduce data.collect;
```

(b) Print Hello World on Scala IDE

```
package hello.world
object hello {
    def main(args: Array[String]) {
        println("Hello World")
    }
}
```

LAB-8

Using RDD & FlatMap count how many time each word appears in a file & write out a list of words whose count is strictly greater than 4 using spark

```
val textFile = sc.textFile("/home/wc.txt");
val counts = textFile.flatMap(line.split(" ")).map(word => (word,1))
    .reduceByKey(_+_)
import scala.collection.immutable.ListMap
val Sorted = ListMap(counts.collect.sortWith(_._2 > _._2):_*)
println(Sorted)
for((K,V) <- sorted) if(V>4)
{
    print(K+", ")
    print(V)
    println()
}
```

Output

```
Spark 7
Hadoop 14
Simple 5
```