Map-Reduce

\*hadoop runs on cloudera operating system. Same os like ubuntu

Few cloudera terminal commands

1) To create a new file-

2) To create a new directory or folder-

3) To change directory-

4) To open a file- application filename.extension

Ex-gedit abc.txt

5) To delete a file from hadoop database-

6) To put a file on hadoop server-

7) To execute a jar file-

Consider a text file sales.txt stored on hadoop server-

2013,hp,monitor,60

2013,hcl,keyboard,55

2013,hp,keyboard,43

2013,intex,mouse,22

2014,hcl,keyboard,23

2014,intex,monitor,67

2014,hcl,mouse,45

2015,intex,keyboard,98

2015,longitech,printer,11

2015,frontech,cpu,24

2015,hp,mouse,41

2016,longitech,mouse,33

2016,frontech,printer,100

Hadoop do not have java data types like int or String they have classes

int=IntWritable

String=Text

long=LongWritable

CLASS 1🡪

MapReduce

MapReduce basically works on 3 classes-

1. Mapper
2. Reducer
3. Driver

***Mapper-***

This class receives the input from the provided text file.

Ex- 2013,hp,monitor,60

2013,hcl,keyboard,55

2014,hcl,keyboard,23

2014,hp,mouse,99

Mapper works on the concept of Key and Value

\*Key is used for sorting purpose

(Consider brand as a key and sales as value)

At first we get the input in mapper file like-

Key value

1. 2013,hp,monitor,60

58fdhs 2013,hcl,keyboard,55

Data is processed in a way like-

1. Mapper executes
2. Data shuffles
3. Data is sorted
4. Data is combined
5. Reducer comes in action

Output of mapper after all these steps be like be like

Key value

hp (60,50,30) All the sales related to hp

hcl (43,22) All the sales related to hcl

frontech (22,56,11,33) All the sales related to ftech

longitech (53,27) All the sales related to ltech

***ADD CODE PRINT 1***

***Reducer-***

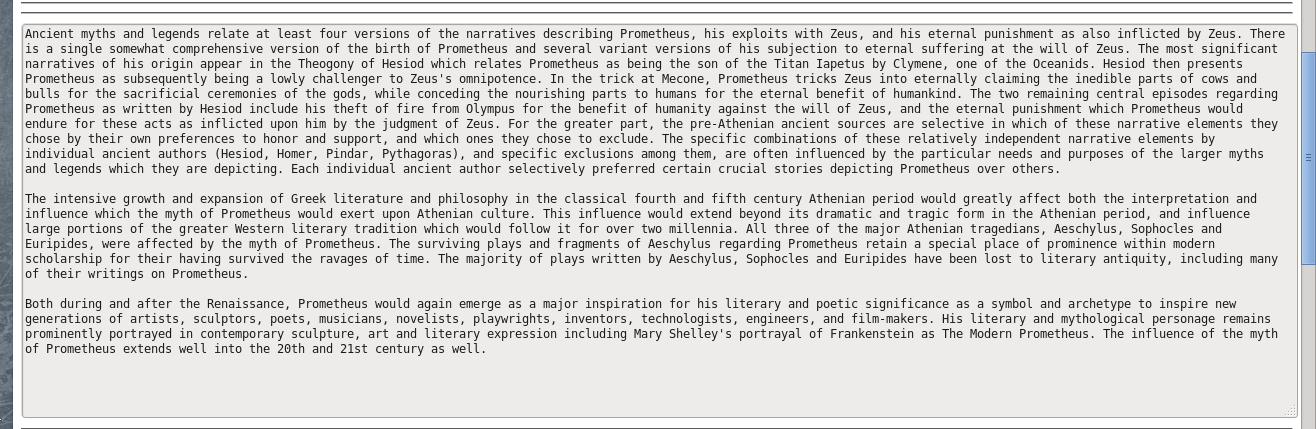
Reducer processing be like

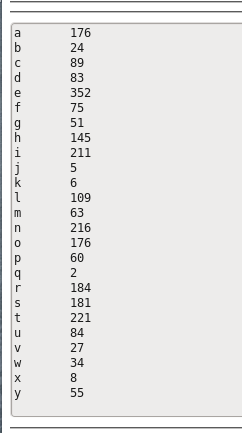
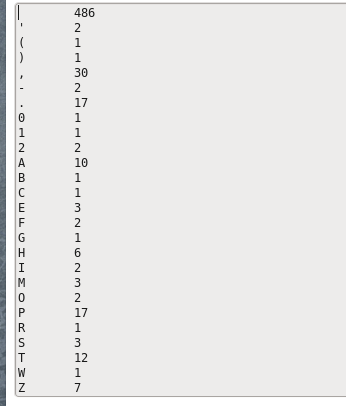
1. Value is in iterable form so can be manipulated to get a desired output.

***Driver-***

Driver class is written in order to configure all the setting of mapper and reducer class and run them in perfect order for a desired output.

PROGRAM 1

PROGRAM 2🡪



***MAPPER***

**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 mprcount extends Mapper<LongWritable, Text, Text, IntWritable> {**

**@Override**

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

**String na=value.toString();**

**char x[]=na.toCharArray();**

**for(int i=0;i<na.length();i++)**

**{**

**Text tx=new Text(Character.toString(x[i]));**

**IntWritable iwr=new IntWritable(1);**

**context.write(tx,iwr);**

**}**

**}}**

***REDUCER***

**import java.io.IOException;**

**import org.apache.hadoop.io.IntWritable;**

**import org.apache.hadoop.io.Text;**

**import org.apache.hadoop.mapreduce.Reducer;**

**public class redcount extends Reducer<Text, IntWritable, Text, IntWritable> {**

**@Override**

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

**int sum=0;**

**for(IntWritable i:values)**

**sum=sum+i.get();**

**context.write(key,new IntWritable(sum));**

**}}**

***DRIVER***

**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 java.io.IOException;**

**@SuppressWarnings("unused")**

**public class dricount {**

**@SuppressWarnings("deprecation")**

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

**if (args.length != 2)**

**{ System.out.printf("Usage: StubDriver <input dir> <output dir>\n");**

**System.exit(-1);}**

**Job job = new Job();**

**job.setJobName("Stub Driver");**

**job.setMapperClass(mprcount.class); //define the mapper class**

**job.setReducerClass(redcount.class); //define the reducer class**

**job.setJarByClass(dricount.class); //define the driver class**

**job.setOutputKeyClass(Text.class);**

**job.setOutputValueClass(IntWritable.class);**

**Path p1=new Path(args[0]);**

**FileInputFormat.addInputPath(job,p1);**

**Path p2=new Path(args[1]);**

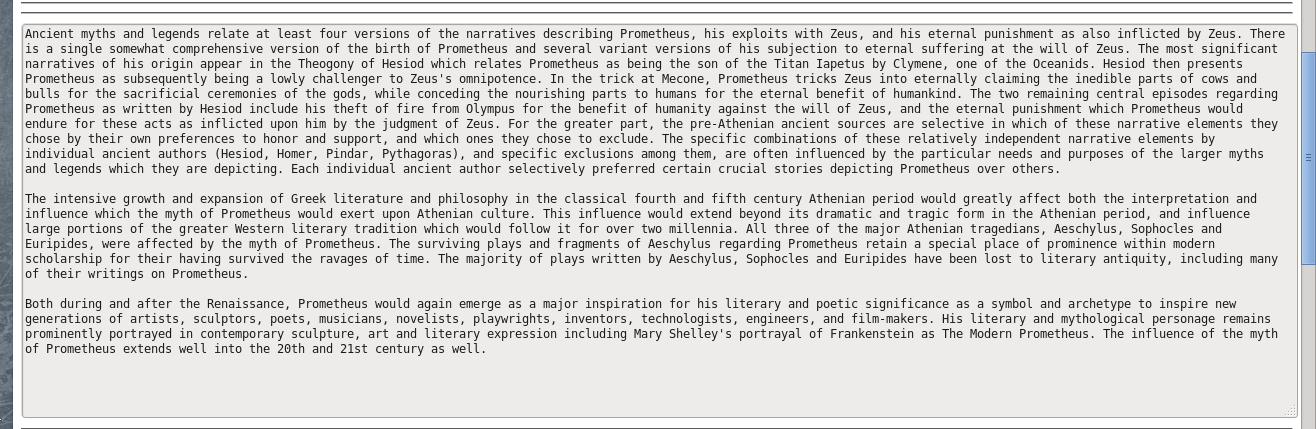
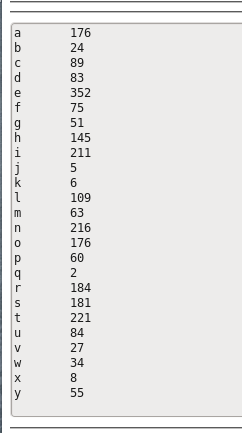
**FileOutputFormat.setOutputPath(job,p2);**

**boolean success = job.waitForCompletion(true);**

**System.exit(success ? 0 : 1);**

**}}**

PROGRAM 3🡪



***REDUCER***

**import java.io.IOException;**

**import org.apache.hadoop.io.IntWritable;**

**import org.apache.hadoop.io.Text;**

**import org.apache.hadoop.mapreduce.Reducer;**

**public class redcount extends Reducer<Text, IntWritable, Text, IntWritable> {**

**@Override**

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

**int sum=0;**

**for(IntWritable i:values) {**

**sum=sum+i.get(); }**

**if(96<(int)d[0] && (int)d[0]<123)**

**{ context.write(key,new IntWritable(sum)); }**

**}}**

PROGRAM 4🡪