PRACTICAL-5

AIM:

To design and implement MapReduce algorithms to take a very large file of integers and produce as output:

a) The largest integer

b) The average of all the integers.

c) The same set of integers, but with each integer appearing only once.

d) The count of the number of distinct integers in the input.

IMPLEMENTATION:

We will follow the below mentioned steps for all the java files.

- Firstly, check whether Hadoop is installed or not.
- Then, make sure that java compiler is running correctly.
- Now, create a folder and a text file for the input.
- Also, create another folder to store java classes files.
- Now, set Hadoop classpath environment variable.
- Create a directory on HDFS.
- Upload the input file to that directory.
- Change the directory to the one where all the files are located.
- Then, compile the java code.
- Class files are generated in the classes folder.
- Put the output files in one jar files.
- Run the jar file on Hadoop
- Check the output

Following commands will be used:

- Export HADOOP_CLASSPATH=$(Hadoop classpath)
- Javac -classpath ${HADOOP_CLASSPATH} -d <path-to-local-classes-folder> <path-to-wordcount-java-file>
- Jar -cvf <any-file-name.jar> -C <path-to-class-folder-in-local>
- Hadoop jar <path-to-jar-file> <classname> <path-to-hdfs-input_data> <path-to-hdfs-output>

PROGRAM CODE:

Largest Number:

```
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class IntMax {

  public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text("max");
    private int max = Integer.MIN_VALUE;

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        //word.set(itr.nextToken());
```

```
      //context.write(word, one);
          int temp = Integer.parseInt(itr.nextToken());
          if(temp > max)
            max = temp;
    }

    context.write(word, new IntWritable(max));
  }
}

public static class IntSumReducer
     extends Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int max = Integer.MIN_VALUE;
    for (IntWritable val : values) {
      //sum += val.get();
          if(val.get() > max)
            max = val.get();
    }
    result.set(max);
    context.write(key, result);
  }
}

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

```
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Max int");
    job.setJarByClass(IntMax.class);
    job.setMapperClass(TokenizerMapper.class);
    //job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.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);
  }
}
```

INPUT:

0,9,3,2,1,8,5,6,7

OUTPUT:

```
admin123@admin123-VirtualBox:/usr/local/hadoop/sbin$ hdfs dfs -ls /prac5_a/Outp
ut3
```

```
admin123@admin123-VirtualBox:/usr/local/hadoop/sbin$ hdfs dfs -cat /prac5_a/Out
put3/part-r-00000
max     9
```

PROGRAM CODE:

Average of integers:

```
import java.io.IOException;
import java.util.StringTokenizer;
import java.io.DataInput;
import java.io.DataOutput;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

class Custom implements Writable {
    private int sum;
    private int count;

    public Custom()
    {
        sum = 0;
        count = 0;
    }

    public void write(DataOutput dataOutput) throws IOException
```

```
{
        dataOutput.writeInt(sum);
        dataOutput.writeInt(count);
}

public void readFields(DataInput dataInput) throws IOException
{
        sum = dataInput.readInt();
        count = dataInput.readInt();
}

public void setSum(int value)
{
        sum = value;
}

public int getSum()
{
        return sum;
}

public void setCount(int value)
{
        count = value;
}

public int getCount()
{
        return count;
```

```
    }

    public String toString()
    {
            return "("+sum+","+count+")";
    }
}

public class IntAvg {

  public static class TokenizerMapper
       extends Mapper<Object, Text, Text, Custom>{

    //private final static IntWritable one = new IntWritable(1);
    private Text word = new Text("avg");
    private Custom obj = new Custom();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      int sum = 0;
      int count = 0;

      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        //word.set(itr.nextToken());
        //context.write(word, one);
            int temp = Integer.parseInt(itr.nextToken());
            sum += temp;
            count++;
```

```
        //context.write(new Text(""+temp), new Text("("+sum+","+count+")"));
    }
    obj.setSum(sum);
    obj.setCount(count);
    context.write(word, obj);
    //context.write(new Text("Avg"), new Text("("+sum+","+count+")"));
  }
}

public static class IntSumReducer
     extends Reducer<Text,Custom,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<Custom> values,
                     Context context
                     ) throws IOException, InterruptedException {
    int sum = 0;
    int count = 0;
    for (Custom val : values) {
      sum += val.getSum();
        count += val.getCount();
    }
    result.set((int)(sum/count));
    context.write(key, result);
  }
}


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

```
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Average");
    job.setJarByClass(IntAvg.class);
    job.setMapperClass(TokenizerMapper.class);
    //job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);

    job.setMapOutputValueClass(Custom.class);

    job.setOutputKeyClass(Text.class);
    //job.setOutputValueClass(IntWritable.class);
    job.setOutputValueClass(Text.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

INPUT:

0,9,3,2,1,8,5,6,7, 20,11,12,19,13,18,14,17,15,16

OUTPUT:

```
admin123@admin123-VirtualBox:/usr/local/hadoop/sbin$ hdfs dfs -cat /prac5_b/Out
put19/part-r-00000
avg     10
```

PROGRAM CODE:

Unique Number in the set:

```
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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class UniqueSet {

  public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

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

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
```

```
    }
  }
}

public static class UniqueKeys
    extends Reducer<Text,IntWritable,Text,Text> {
  private IntWritable result = new IntWritable();

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

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "Unique Set");
  job.setJarByClass(UniqueSet.class);
  job.setMapperClass(TokenizerMapper.class);
  //job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(UniqueKeys.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);
  }
}
```

INPUT:

1,2,3,4,5,6,7,8,9,3,7,3,8,3,1,5,9,4,2,7,3,6,7,1

OUTPUT:

PROGRAM CODE:

For Count of distinct integers in the input:

```
import java.io.IOException;
import java.util.StringTokenizer;
import java.util.LinkedList;

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.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
```

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

public class UniqueCount {

  public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

    //private final static IntWritable one = new IntWritable(1);
    //private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        context.write(new Text("Count"), new IntWritable(Integer.parseInt(itr.nextToken())));
      }
    }
  }

  public static class UniqueKeys
      extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();
    //private LinkedList intSet = new LinkedList<IntWritable>();
    //private int count = 0;

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      //int sum = 0;
```

```
    int count = 0;
    LinkedList intSet = new LinkedList<Integer>();
    for (IntWritable val : values) {
    //  sum += val.get();
        //int temp = val.get();
            if(!intSet.contains(val.get()))
            {
              count++;
              intSet.add(val.get());
              //context.write(new Text("Added"), val);
            }
            //context.write(key, val);
    }
    //result.set(sum);
    context.write(key, new IntWritable(count));
  }
}

public static void main(String[] args) throws Exception {
  Configuration conf = new Configuration();
  Job job = Job.getInstance(conf, "Unique Count");
  job.setJarByClass(UniqueCount.class);
  job.setMapperClass(TokenizerMapper.class);
  //job.setCombinerClass(IntSumReducer.class);
  job.setReducerClass(UniqueKeys.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);
  }
}
```

INPUT:

0,9,8,7,6,5,4,3,3,2,1,1,2,3,3,4,5,6,7,8,9,0

OUTPUT:

CONCLUSION:

In this practical, I learnt to perform mapreduce algorithms on integers.