**LAB – 4**

Name: Gandevia Keval Dharmeshbhai

Sem: VII

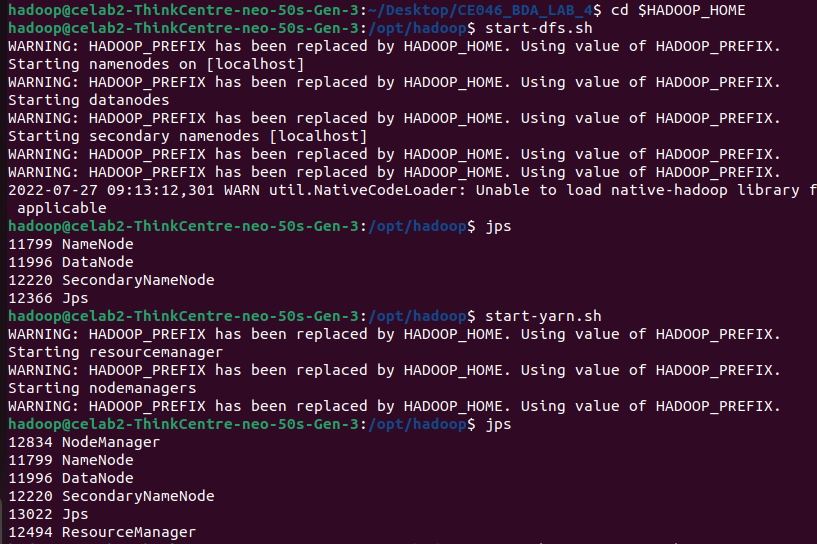
Roll No: CE046

Subject: Big Data and Analytics

**Aim:** Write a map-reduce program to count the frequencies of word from distributed storage source and understand the phases involved in map-reduce programming.

**Q. 1: Wordcount program with map-reduce**

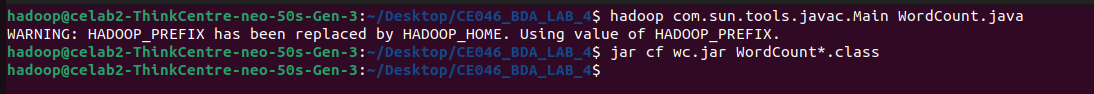
1. Starting the Hadoop server and yarn.



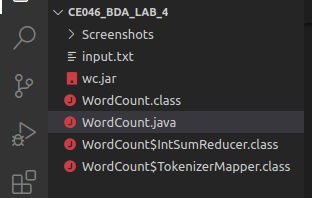
1. Starting history server and verifying using jps. As we can see NameNode, DataNode, and SecodaryNameNode are the processes spawned by the Hadoop server. NodeManager and ResourceManager are the processes spawned by the yarn. Moreover, JobHistoryServer is spawned by the history server.



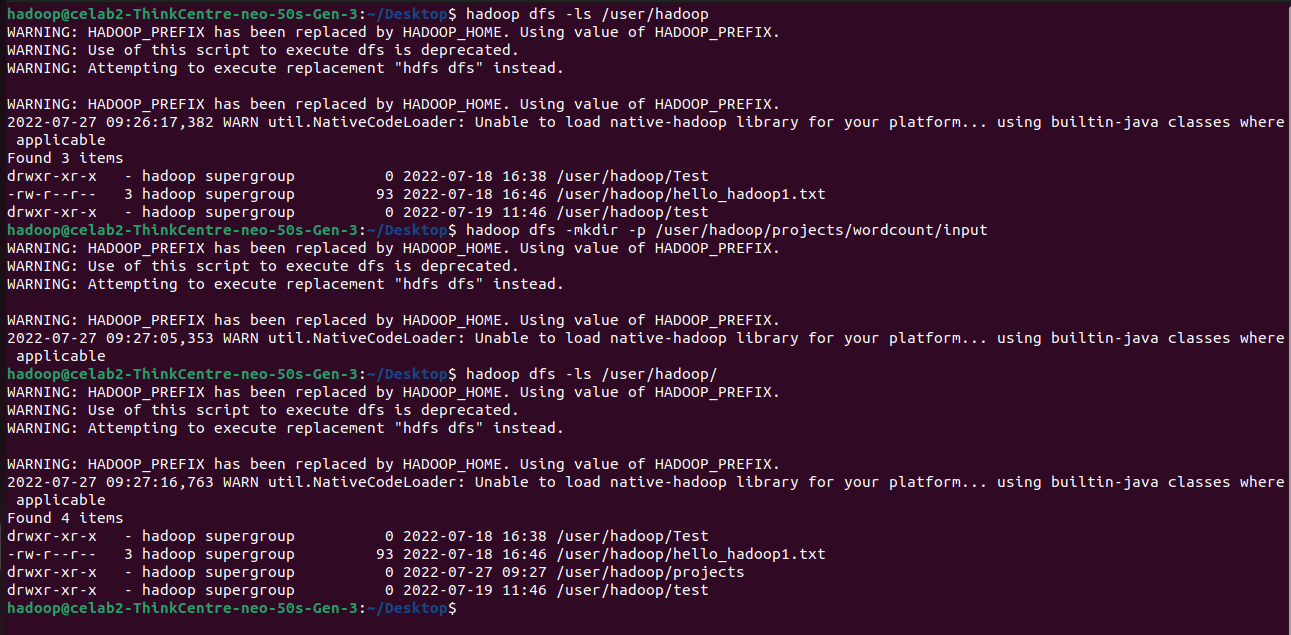
1. Compiling program and creating a jar file.

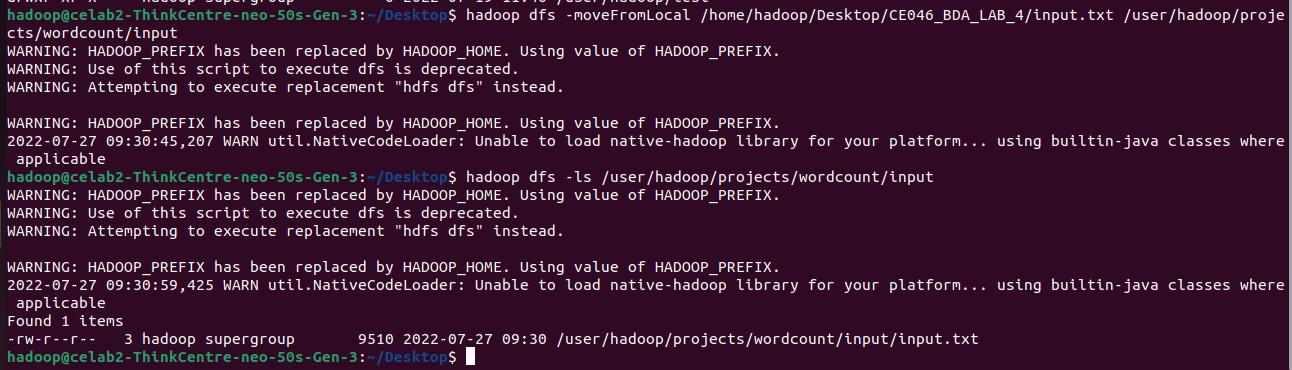


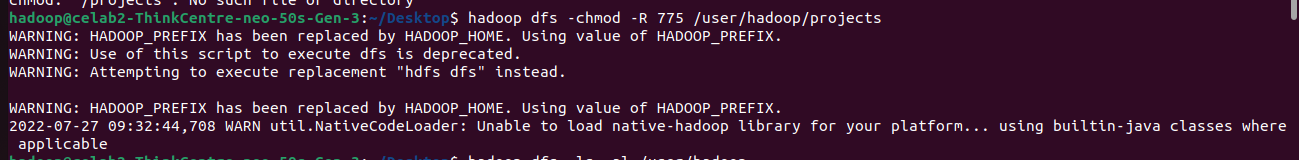
1. Program is complied successfully.

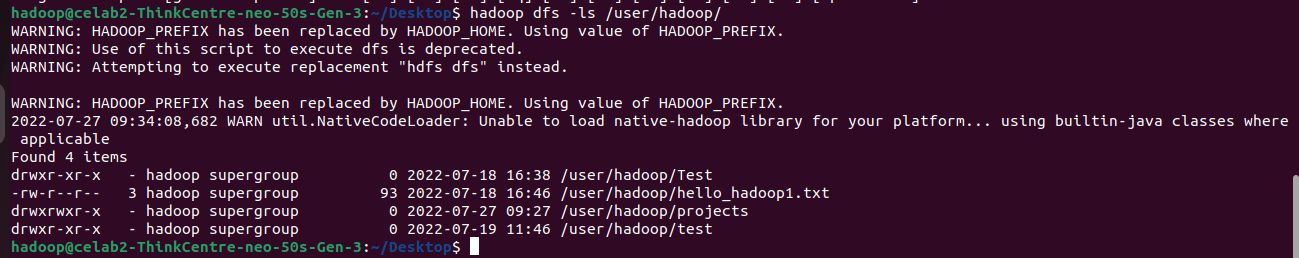


1. Creating directory and uploading a text file into the HDFS.

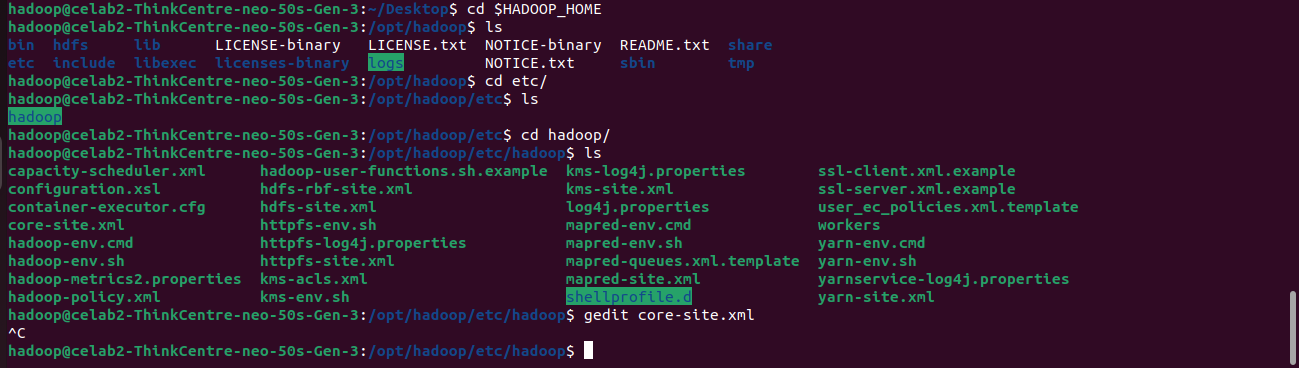








1. Editing configuration files.



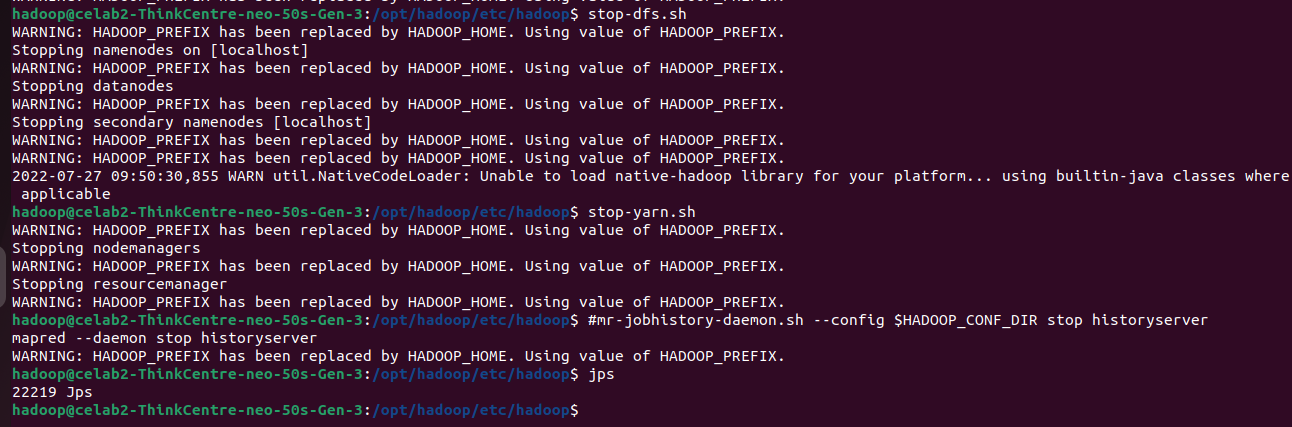


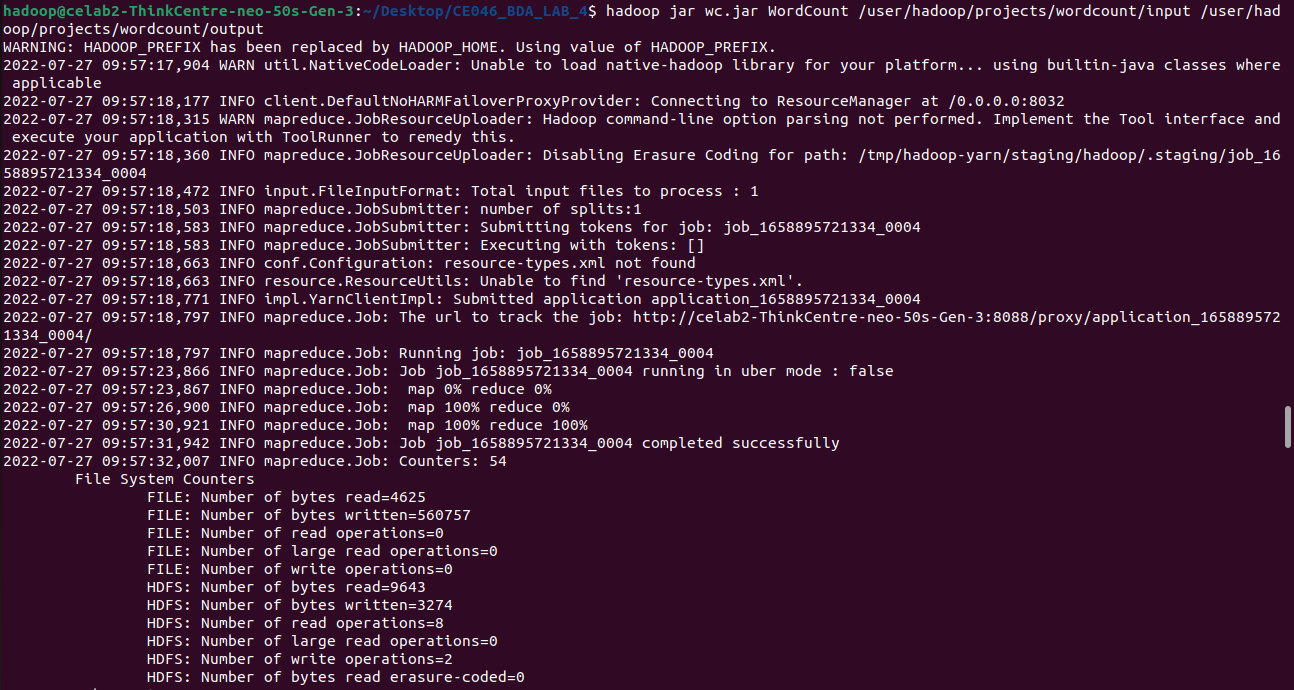




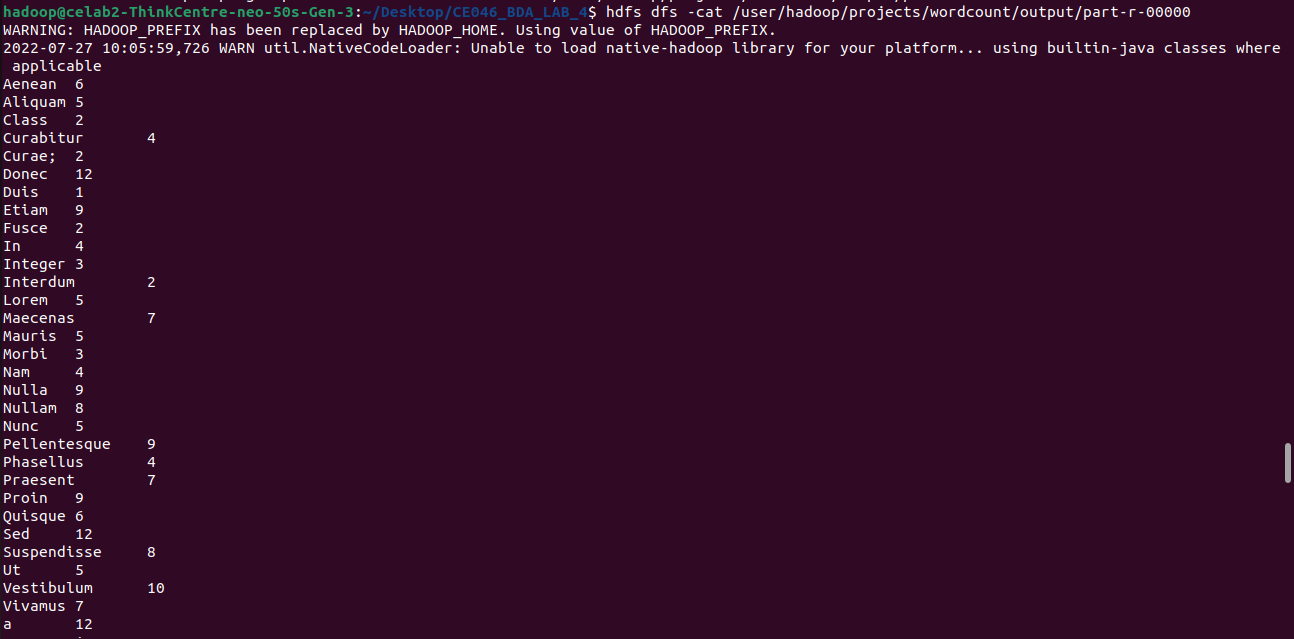


1. Need to stop and restart the server.





1. Output.





**Q. 2: Write map and reduce functions to split the books into the following two categories: (a) Big Books, (b) Small Books. Books which have more than 300 pages should be in the big book category. Books which have less than 300 pages should be in the small book category.**

* **Code:**

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.\*;

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    Book

{

    public static class TokenizerMapper extends Mapper <LongWritable, Text, Text, IntWritable>

    {

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

        private Text word = *new* Text();

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

        {

            StringTokenizer tokenizer = *new* StringTokenizer(value.toString());

            int count = 0;

*while* (tokenizer.hasMoreTokens()) {

*if*(Integer.parseInt(tokenizer.nextToken()) >= 300)

                {

                    word.set("Big Books");

                }

*else*

                {

                    word.set("Small Books");

                }

                context.write(word, one);

            }

        }

    }

    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 sum = 0;

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

                sum += val.get();

            }

            result.set(sum);

            context.write(key, result);

        }

    }

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

    {

        Configuration   conf = *new* Configuration();

        Job     job = Job.getInstance(conf, "book count");

                job.setJarByClass(Book.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]));

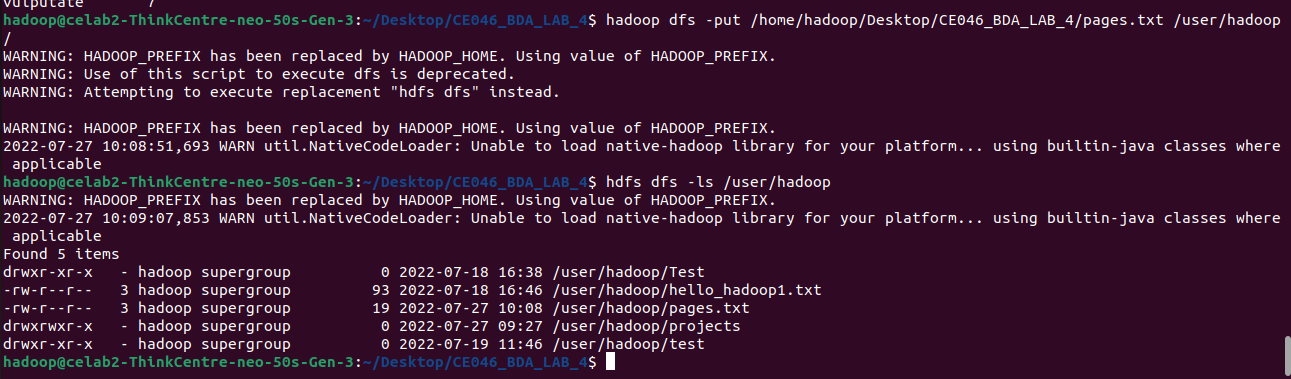
                System. exit  (job.waitForCompletion(true) *?* 0 *:* 1);

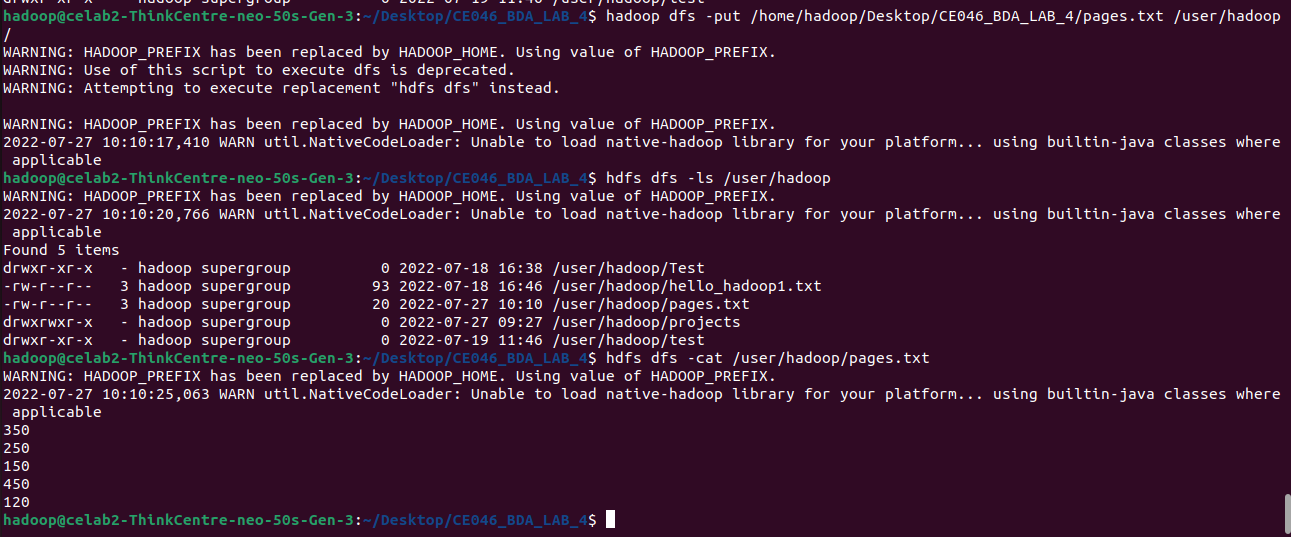
    }

}

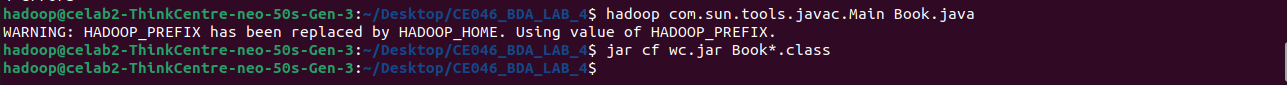
* **Steps:**

1. Putting pages.txt file onto the Hadoop server.





1. Compiling program and creating a jar file.



1. Output.

