**A REPORT ON**

**SMART PARKING SYSTEM**

**BY**

|  |  |
| --- | --- |
| **KAVYA S**  **MOUNIKA L**  **RISHIKA RAO** | **2016CSE289**  **2016CSE299**  **2016CSE339** |

**AT**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING CSE 219 BIG DATA ANALYTICS**

**MINI PROJECT REPORT**



**PRESIDENCY UNIVERSITY, BENGALURU DEC 2019**

**DECLARATION**

I hereby declare that the report entitled “Smart parking system” is submitted to CSE 219 big data analytics lab is carried out by us. The material contained in this report has not been submitted to anywhere else.

|  |  |  |
| --- | --- | --- |
| Name of the student | ID Number | Signature |
| **KAVYA S**  **MOUNIKA L**  **RISHIKA VARSHINI RAO** | **2016CSE289**  **2016CSE299**  **2016CSE239** |  |

Abstract

Smart Parking involves the use of low cost sensors, real-time data and applications.

The goal is to automate and decrease time spent manually on allowance of vehicles inside

any organization. Quick parking in no time. Eliminates /Reduces staff costs--works automatically.Typically installed at shopping malls, airports, hospitals, office buildings, residentialCommunities, etc. Here the data analysis is on the maximum and minimum number of times vehicleentered in a particular day.

Signature of Student:

Guided by

Date:11-12-2019 Dr. R.Sathish Kumar

**TABLE OF CONTENTS**

**1. Declaration**

**2. Abstract**

**3.Introduction to application**

1. **Objective 1 (count number of times a car entered)**
   1. **Description of the data set**
   2. **Screen short of data set**
   3. **3 java files (Mapper, Reducer, Main File)**
   4. **Output screen short with explanation**
2. **Objective 2 (Sorting the output file of first program)**
   1. **Description of the data set**
   2. **Screen short of data set**
   3. **3 java files (Mapper, Reducer, Main File)**
   4. **Output screen short with explanation**
3. **Objective 3 (Extract the maximum and minimum number of times a car entered)**
   1. **Description of the data set**
   2. **Screen short of data set**
   3. **3 java files (Mapper, Reducer, Main File)**
   4. **Output screen short with explanation**
4. **Conclusion**
5. **References**

Flow of your report

**Introduction to application**

**Objective 1 (Title)**

* 1. **Description of the data set**
  2. **Screen short of your data set**
  3. **Copy past the 3 java files (Mapper, Reducer, Main File)**
  4. **Output screen short with explanation**

**Objective 2 (Title)**

* 1. **Description of the data set**
  2. **Screen short of your data set**
  3. **Copy past the 3 java files (Mapper, Reducer, Main File)**
  4. **Output screen short with explanation**

**Objective 3 (Title)**

* 1. **Description of the data set**
  2. **Screen short of your data set**
  3. **Copy past the 3 java files (Mapper, Reducer, Main File)**
  4. **Output screen short with explanation**

**Conclusion**

**References**

**1.Introduction to application**

Large-scale Internet of Things (IoT) deployments are being massively installed within smart cities , and alongside their adoption, there is a concurrent need for advanced processing functionalities to handle the vast amount of data generated by sensor devices and, more importantly, to make these data useful for public administrations and citizens. IoT technology allows monitoring a wide range of physical objects through low-cost and possibly low-power sensing and transmission technologies. Nevertheless, despite the ever-growing interest for IoT, to date, there have been few technical investigations employing data analytics to solve real-world problems in smart cities and especially utilizing IoT data as a basis for new applications. Our focus in this paper is on data analysis tools for smart parking systems, and for our designs, tests and considerations, we use data from a smart parking deployment installed. We design processing tools to extract relevant statistical features from real-life parking data.

 Note that this classification is a key feature for parking managers and also reveals interesting aspects.

 For example, it is possible to label neighborhoods as residential or commercial by just looking at how parking spaces are used.

Therefore, besides managing parking spaces and further smart city applications can be built on top of our classification algorithms, by fusing what we learn with other types of data. The usage of automated instrumentation for on-street parking monitoring has become popular in several cities around the world. The main goal of these systems is to improve the operation efficiency of public parking, which is achieved through the collection of fine-grained, constant and accurate information on parking lot occupancy. With this objective in mind, the collected data are analyzed and provided to the city parking management division through suitable dashboards. On top of this, the availability of real-time parking information also enables new services, providing an improved urban user experience.

Here we have performed operations like;

1. Finding maximum and minimum number of times a car entered.
2. First count the number of times a car entered
3. Sort the data
4. And print the minimum and maximum number of times;

**Objective 1 (count number of times a car entered)**

**Description of the data set:**

This data set contains the information of the date, car id, car entered time and model of the car. Each date contains 20 data.

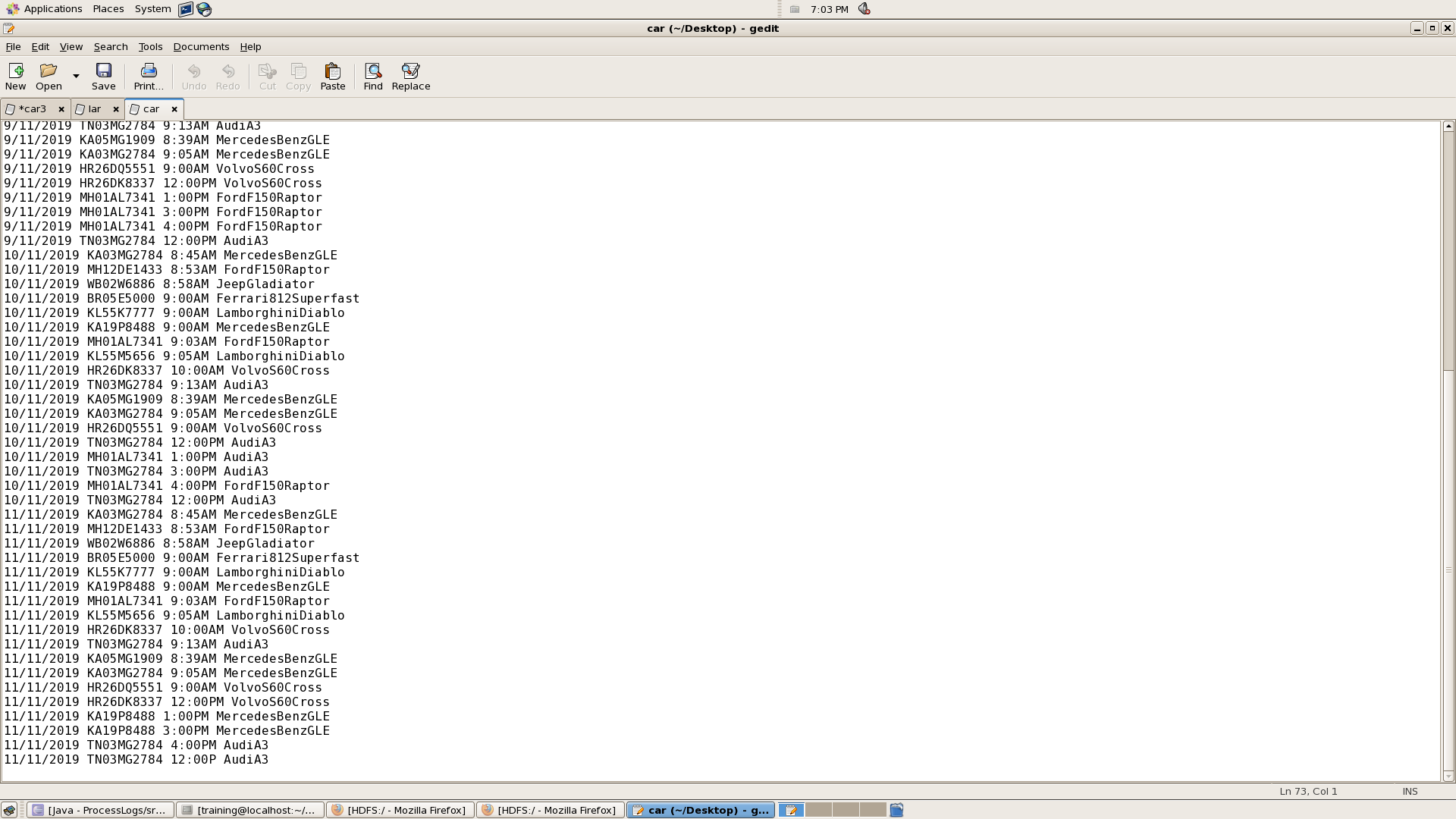
DATE: Date on when the car entered

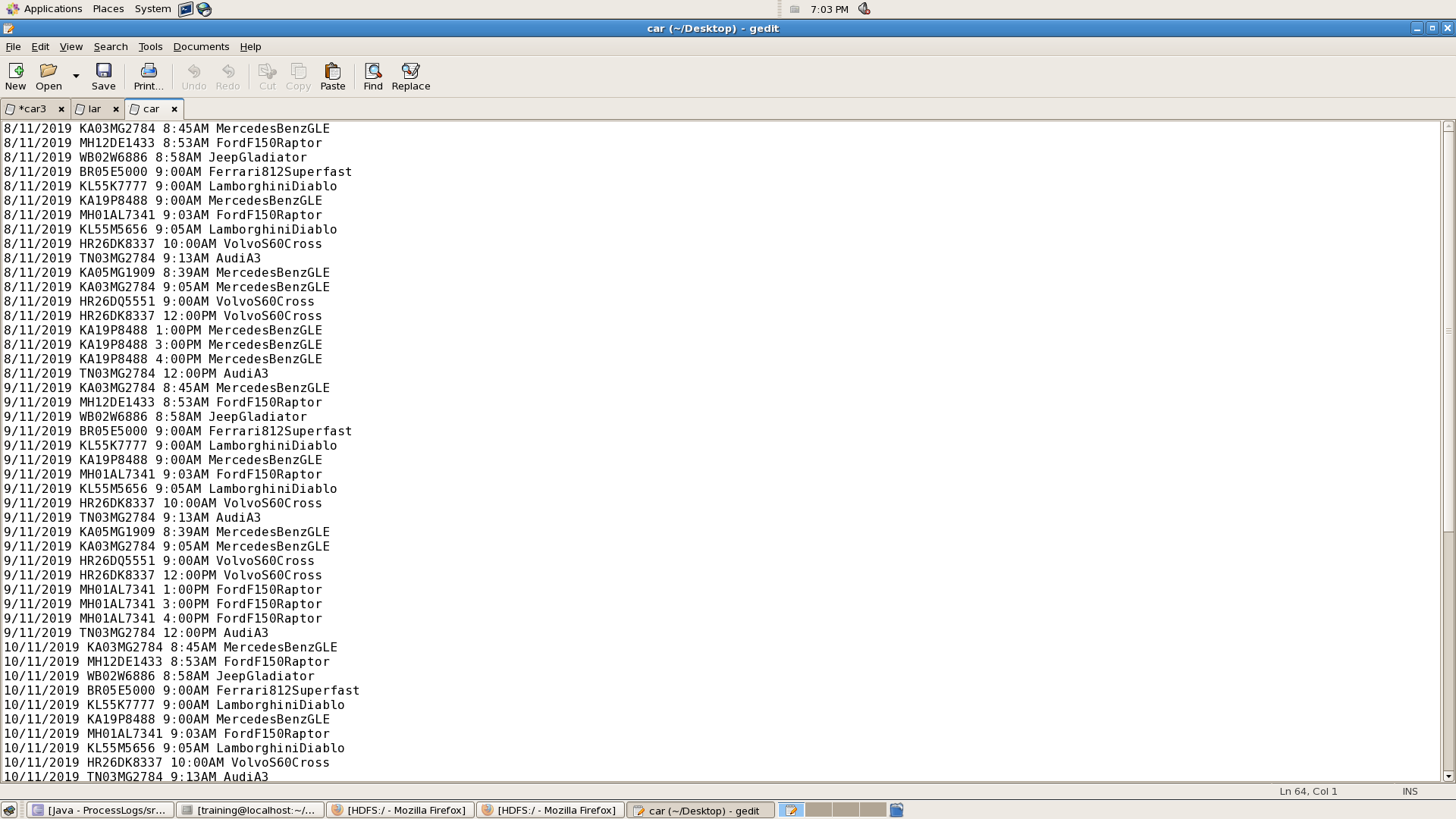
CAR ID: The unique id of the car is stored in the data

Time: Time on when the car entered

Model: The model od the car is taken

**Screen short of our data set:**

****

****

**Java files (Mapper, Reducer, Main File):**

**wordmapper.class:**

**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.mapred.MapReduceBase;

**import** org.apache.hadoop.mapred.Mapper;

**import** org.apache.hadoop.mapred.OutputCollector;

**import** org.apache.hadoop.mapred.Reporter;

**public** **class** wordmapper **extends** MapReduceBase **implements** Mapper<LongWritable,Text,Text,IntWritable>

{

//private Text word=new Text();

**public** **void** map(LongWritable key, Text value,OutputCollector<Text, IntWritable> output, Reporter r)

**throws** IOException {

String s =value.toString();

String[] items=s.split(" ");

String stock=items[1];

output.collect(**new** Text(stock), **new** IntWritable(1));

}

}

**wordreducer.class:**

**import** java.io.IOException;

**import** java.util.Iterator;

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

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

**import** org.apache.hadoop.mapred.MapReduceBase;

**import** org.apache.hadoop.mapred.OutputCollector;

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

**import** org.apache.hadoop.mapred.Reporter;

**public** **class** wordreducer **extends** MapReduceBase **implements** Reducer<Text,IntWritable,Text,IntWritable>

{

**public** **void** reduce(Text key, Iterator<IntWritable> values,

OutputCollector<Text, IntWritable> output, Reporter r)

**throws** IOException {

**int** count=0;

**while**(values.hasNext())

{

IntWritable i= values.next();

count+= i.get();

}

output.collect(key, **new** IntWritable(count));

}

}

**WORDCOUNT.CLASS:**

**import** org.apache.hadoop.conf.Configured;

**import** org.apache.hadoop.fs.Path;

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

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

**import** org.apache.hadoop.mapred.FileInputFormat;

**import** org.apache.hadoop.mapred.FileOutputFormat;

**import** org.apache.hadoop.mapred.JobClient;

**import** org.apache.hadoop.mapred.JobConf;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

**public** **class** wordcount **extends** Configured **implements** Tool {

@Override

**public** **int** run(String[] args) **throws** Exception {

**if**(args.length<2)

{

System.*out*.println("Plz Give Input Output Directory Correctly");

**return** -1;

}

JobConf conf = **new** JobConf(wordcount.**class**);

FileInputFormat.*setInputPaths*(conf,**new** Path(args[0]));

FileOutputFormat.*setOutputPath*(conf, **new** Path(args[1]));

conf.setMapperClass(wordmapper.**class**);

conf.setReducerClass(wordreducer.**class**);

conf.setMapOutputKeyClass(Text.**class**);

conf.setMapOutputValueClass(IntWritable.**class**);

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

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

JobClient.*runJob*(conf);

**return** 0;

}

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

{

**int** exitcode = ToolRunner.*run*(**new** wordcount(), args);

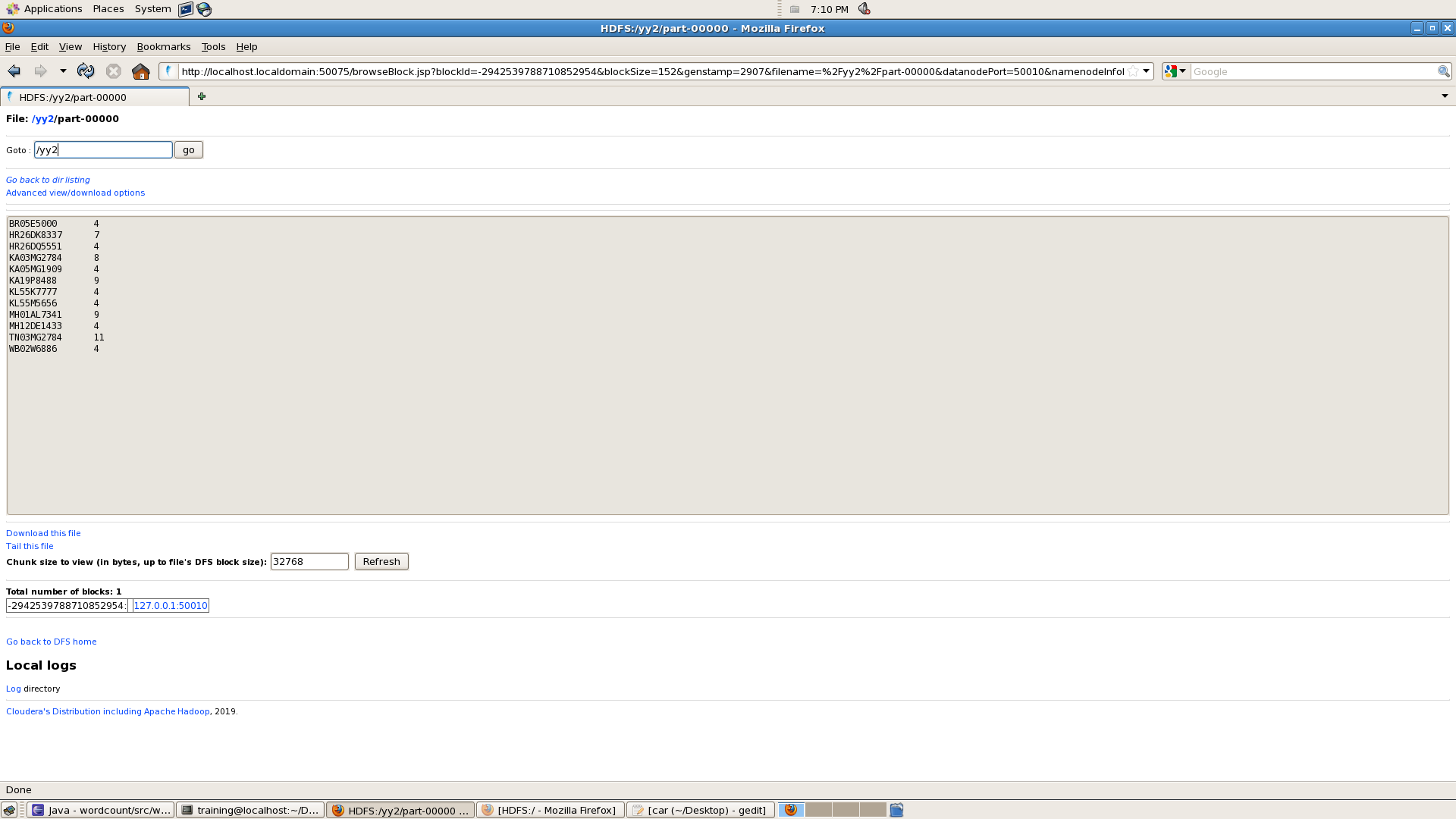
System.*exit*(exitcode);

}

}

**Output screen short with explanation:**

The below image is output for the above mapreduce program where it gives the count of car entered**.**

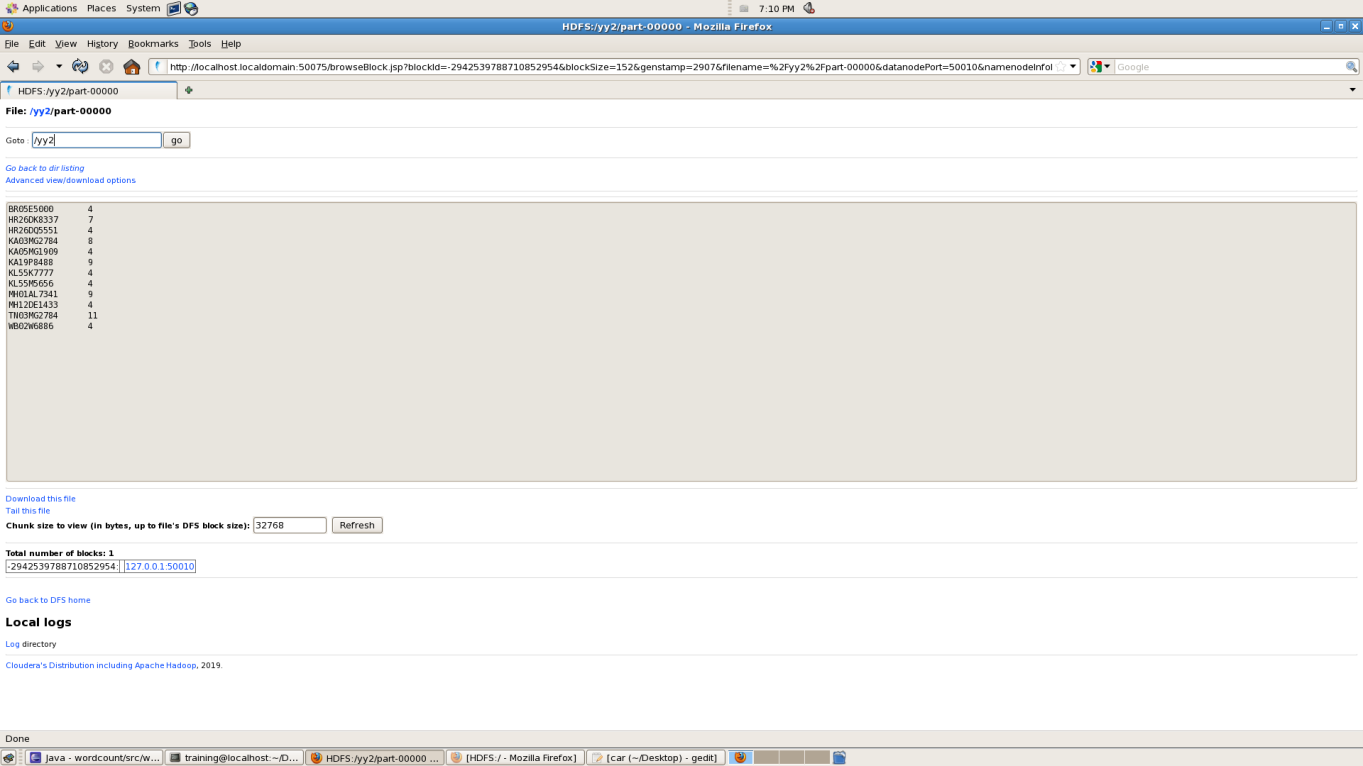


**Objective 2 (Sorting the output file of first program)**

1. **Description of the data set**

The output of the above dataset that is “count” is taken as an input dataset for program

1. **Screen short of your data set**



1. **java files (Mapper, Reducer, Main File):**

**maxcloseprice.class:**

**import** org.apache.hadoop.fs.Path;

**import** org.apache.hadoop.conf.Configured;

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

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

**import** org.apache.hadoop.mapred.FileInputFormat;

**import** org.apache.hadoop.mapred.FileOutputFormat;

**import** org.apache.hadoop.mapred.JobClient;

**import** org.apache.hadoop.mapred.JobConf;

**import** org.apache.hadoop.util.Tool;

**import** org.apache.hadoop.util.ToolRunner;

**public** **class** MaxClosePrice **extends** Configured **implements** Tool {

**public** **int** run(String[] args) **throws** Exception {

**if**(args.length<2)

{

System.*out*.println("Plz Give Input Output Directory Correctly");

**return** -1;

}

JobConf conf = **new** JobConf(MaxClosePrice.**class**);

FileInputFormat.*setInputPaths*(conf,**new** Path(args[0]));

FileOutputFormat.*setOutputPath*(conf, **new** Path(args[1]));

conf.setMapperClass(MaxClosePriceMapper.**class**);

conf.setReducerClass(MaxClosePriceReducer.**class**);

conf.setMapOutputKeyClass(Text.**class**);

conf.setMapOutputValueClass(IntWritable.**class**);

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

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

JobClient.*runJob*(conf);

**return** 0;

}

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

**int** exitcode = ToolRunner.*run*(**new** MaxClosePrice(), args);

System.*exit*(exitcode);

}

}

**maxclosepricemapper.class:**

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class MaxClosePriceReducer extends MapReduceBase implements Reducer<Text,IntWritable,IntWritable,Text>

{

private int max\_temp=Integer.MIN\_VALUE;

private int temp=0;

@Override

public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<IntWritable,Text> output, Reporter r) throws IOException {

while(values.hasNext()){

IntWritable i=values.next();

temp=i.get();

if(temp>max\_temp){

max\_temp=temp;

}

}

output.collect(new IntWritable(max\_temp),key);

}

}

**Maxclosepricereducer.class**:

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

import org.apache.hadoop.mapred.Reporter;

public class MaxClosePriceReducer extends MapReduceBase implements Reducer<Text,IntWritable,IntWritable,Text>

{

private int max\_temp=Integer.MIN\_VALUE;

private int temp=0;

@Override

public void reduce(Text key, Iterator<IntWritable> values,OutputCollector<IntWritable,Text> output, Reporter r) throws IOException {

while(values.hasNext()){

IntWritable i=values.next();

temp=i.get();

if(temp>max\_temp){

max\_temp=temp;

}

}

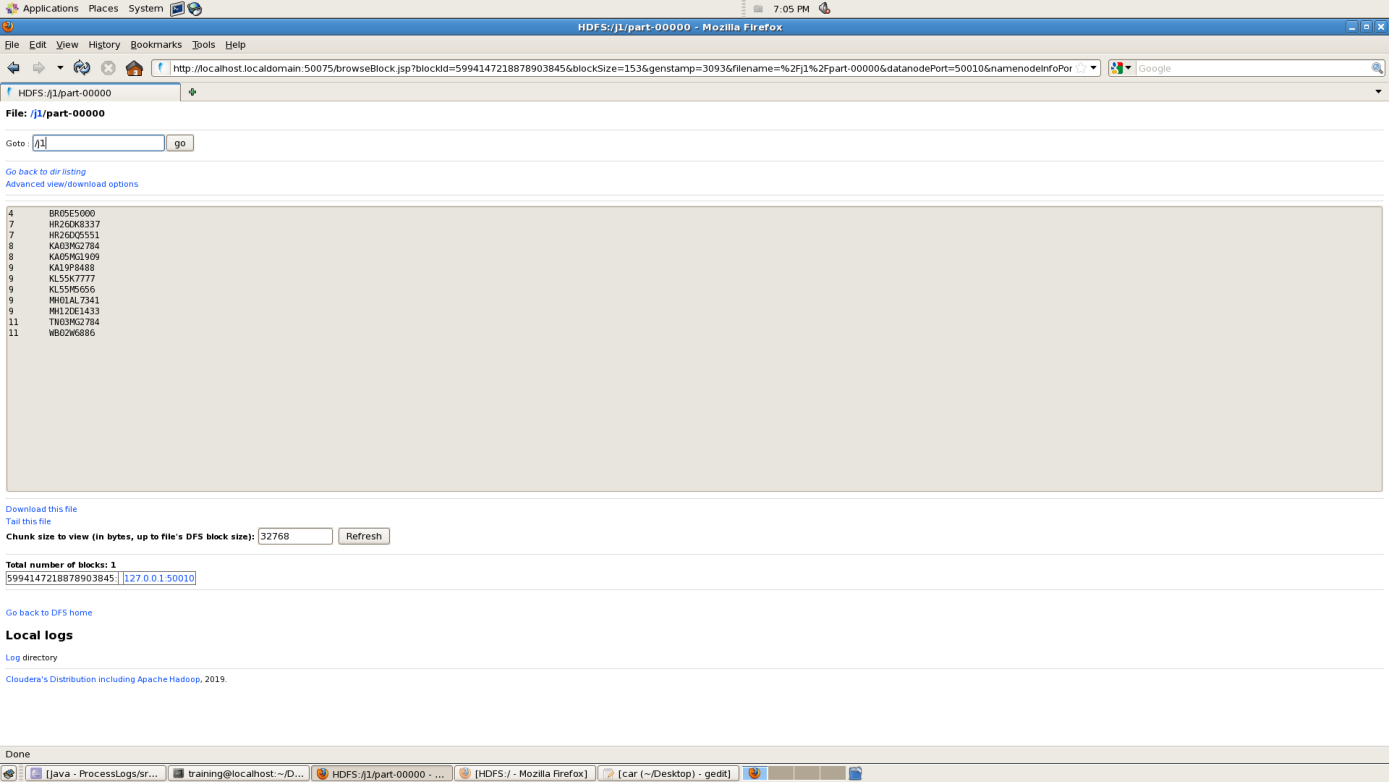
output.collect(new IntWritable(max\_temp),key);

}

}

1. **Output screen short with explanation:**

Here the output has been sorted according to the number of times a car entered.

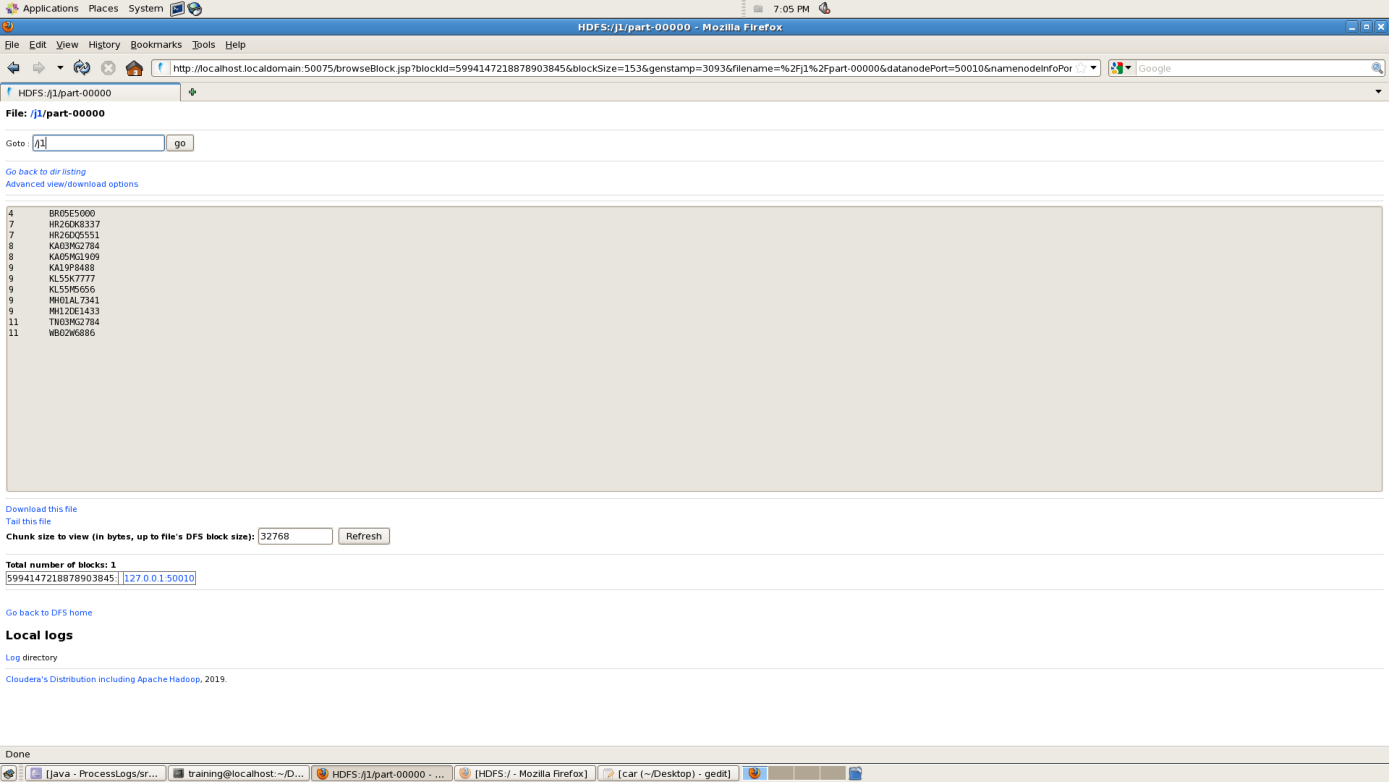
****

**Objective 3 (Extract the maximum and minimum number of times a car entered)**

1. **Description of the data set**

The data set for this is the output data of the above program that is “sorting”.

1. **Screen short of our data set**

****

1. **java files (Mapper, Reducer, Main File)**

**proceeslogs.class:**

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

public class ProcessLogs extends Configured implements Tool {

@Override

public int run(String[] args) throws Exception {

if(args.length<2)

{

System.out.println("Plz Give Input Output Directory Correctly");

return -1;

}

JobConf conf = new JobConf(ProcessLogs.class);

FileInputFormat.setInputPaths(conf,new Path(args[0]));

FileOutputFormat.setOutputPath(conf, new Path(args[1]));

conf.setMapperClass(LogMapper.class);

conf.setReducerClass(LogReducer.class);

conf.setMapOutputKeyClass(Text.class);

conf.setMapOutputValueClass(IntWritable.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);

return 0;

}

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

{

int exitcode = ToolRunner.run(new ProcessLogs(), args);

System.exit(exitcode);

}

}

**logmapper.class:**

**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.mapred.MapReduceBase;

**import** org.apache.hadoop.mapred.Mapper;

**import** org.apache.hadoop.mapred.OutputCollector;

**import** org.apache.hadoop.mapred.Reporter;

**public** **class** LogMapper **extends** MapReduceBase **implements** Mapper<LongWritable,Text,Text,IntWritable>

{

**public** **void** map(LongWritable key, Text values,

OutputCollector<Text, IntWritable> output, Reporter r)

**throws** IOException {

String line = values.toString();

String[] w=line.split("\t");

**int** sal=Integer.*parseInt*(w[0]);

String name=w[1];

output.collect(**new** Text(name), **new** IntWritable(sal));

}

}

**logreducer.class:**

**import** java.io.IOException;

**import** java.util.Iterator;

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

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

**import** org.apache.hadoop.mapred.MapReduceBase;

**import** org.apache.hadoop.mapred.OutputCollector;

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

**import** org.apache.hadoop.mapred.Reporter;

**public** **class** LogReducer **extends** MapReduceBase **implements** Reducer<Text,IntWritable,IntWritable,Text>

{

**int** arr[]= **new** **int**[50];

**int** j=0;

**public** **void** reduce(Text key, Iterator<IntWritable> values,OutputCollector<IntWritable,Text> output, Reporter r) **throws** IOException {

**while**(values.hasNext()){

IntWritable i=values.next();

arr[j]=i.get();

j++;}

if(i.get()==4){

**if**(j==0){

output.collect(**new** IntWritable(arr[0]),key);

}

if(i.get()==11){

output.collect(**new** IntWritable(arr[11]),key);

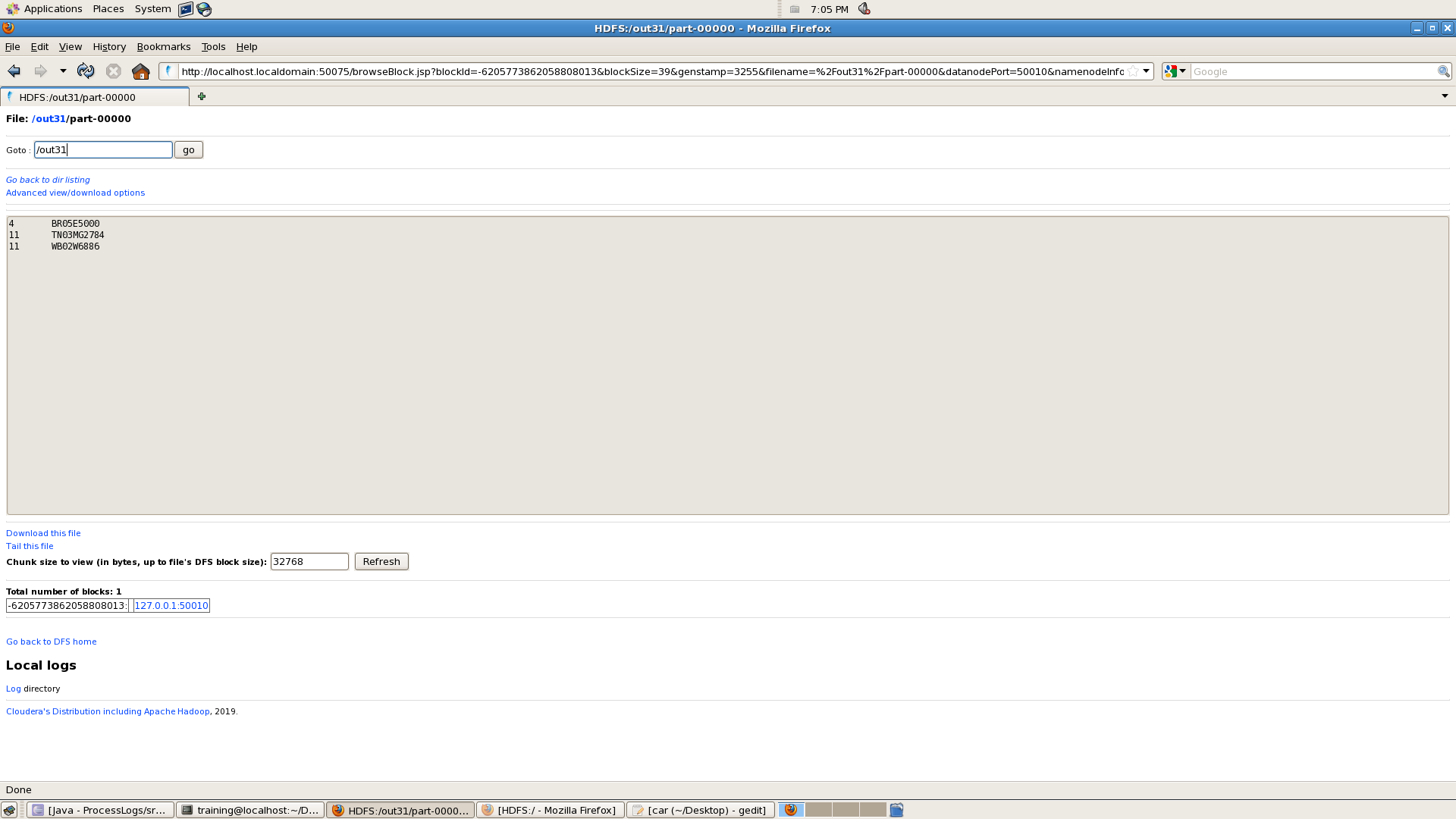
}

}}

}

**d.Output screen short with explanation:**

Here the car id number that as entered minimum and maximum number of times and with there count the output is generated.



**Conclusion:**

In this paper we have investigated classification scheme for smart parking system application. Real data from a commercial deployment were used to understand the peculiarity of real world parking events. The present work is a step towards big data for smart parking system. We in fact believe that parking traces, besides meaningful to parking applications. May also reveal interesting facts about mobility, help implement parking management solutions.

**References:**

[***https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5087364/***](https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5087364/)

***2)***

***https://codereview.stackexchange.com › questions › calculate-min-max-av/***

***3)***

***https://groups.google.com › topic › mongodb-user***

***4)***

***https://sunilmistri.wordpress.com › 2015/02/13 › mapreduce-example-for-/***

***5)***

***<https://www.npntraining.com › blog › map-reduce-min-max-temperature-...>***