

VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS

Grade **A++** Accredited Institution by NAAC

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;
Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE

An ISO 9001:2015 Certified Institution

SCHOOL OF ENGINEERING & TECHNOLOGY

BTECH Programme: **AIDS-A**

Course Title: **Big Data Analytics Lab**

Course Code: **AIDS306P**

Submitted To

Dr. Lakshita Aggarwal

Submitted By

Name: Parina Garg

Enrollment No:02917711922



VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS

Grade A++ Accredited Institution by NAAC

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;

Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE

An ISO 9001:2015 Certified Institution

SCHOOL OF ENGINEERING & TECHNOLOGY

VISION OF INSTITUTE

To be an educational institute that empowers the field of engineering to build a sustainable future by providing quality education with innovative practices that supports people, planet and profit.

MISSION OF INSTITUTE

To groom the future engineers by providing value-based education and awakening students' curiosity, nurturing creativity and building capabilities to enable them to make significant contributions to the world.



VIVEKANANDA INSTITUTE OF PROFESSIONAL STUDIES - TECHNICAL CAMPUS

Grade A++ Accredited Institution by NAAC

NBA Accredited for MCA Programme; Recognized under Section 2(f) by UGC;
Affiliated to OGCIP University, Delhi; Recognized by Bar Council of India and AICTE

Affiliated to GGSIP University, Delhi; Recognized by Bar Council of India and AICTE
An ISO 9001-2015 Certified Institution

An ISO 9001:2015 Certified Institution
SCHOOL OF ENGINEERING & TECHNOLOGY

INDEX

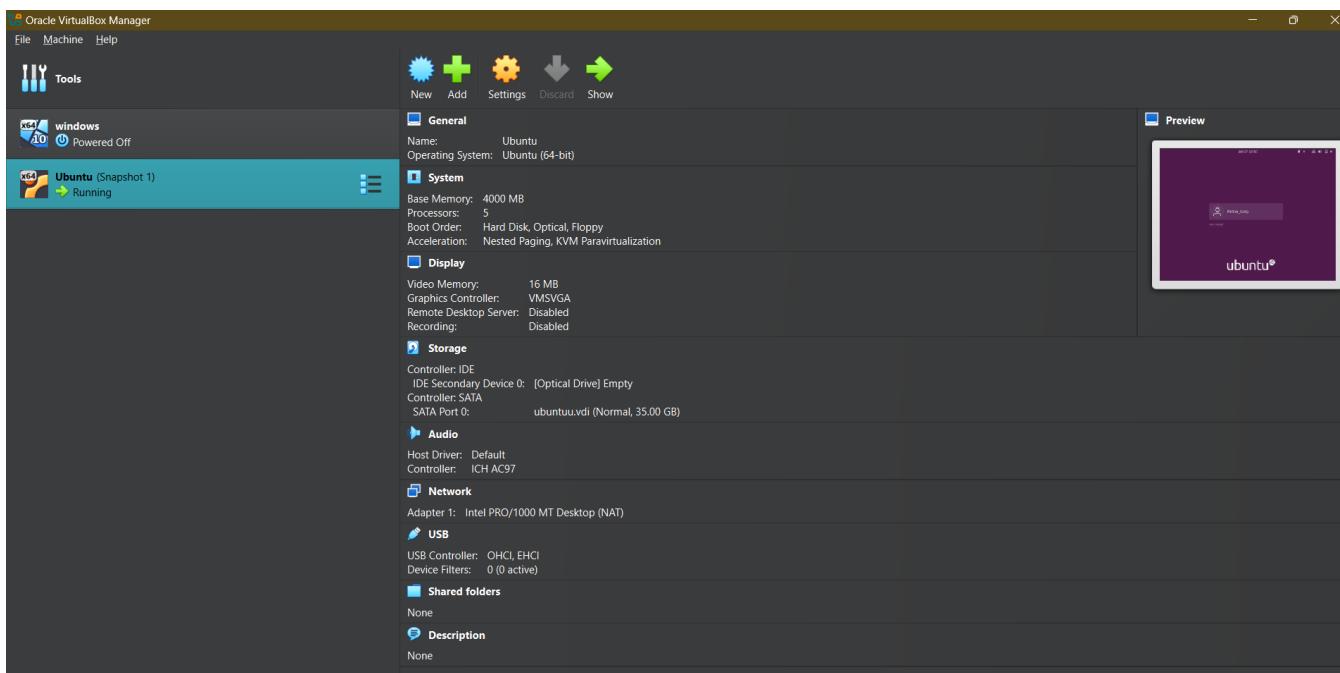
EXPERIMENT-1

Title :- Install Apache Hadoop.

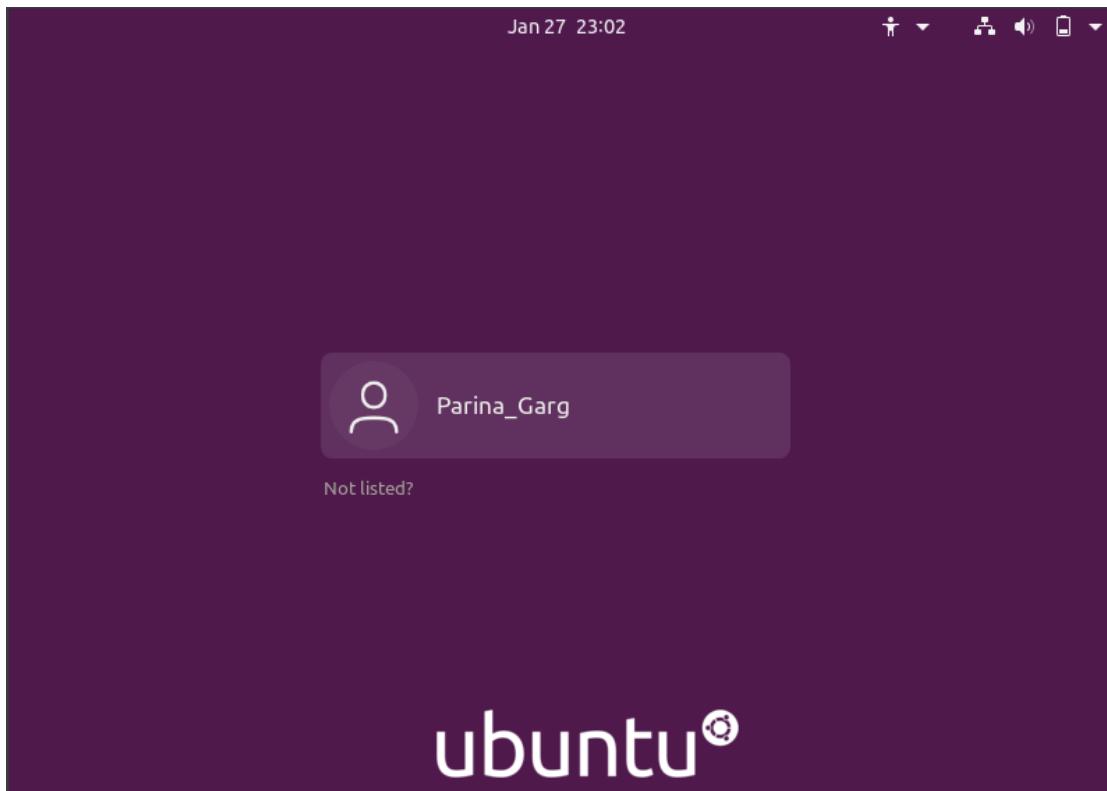
- **Introduction -**

- Setting up Ubuntu on Virtual Machine:

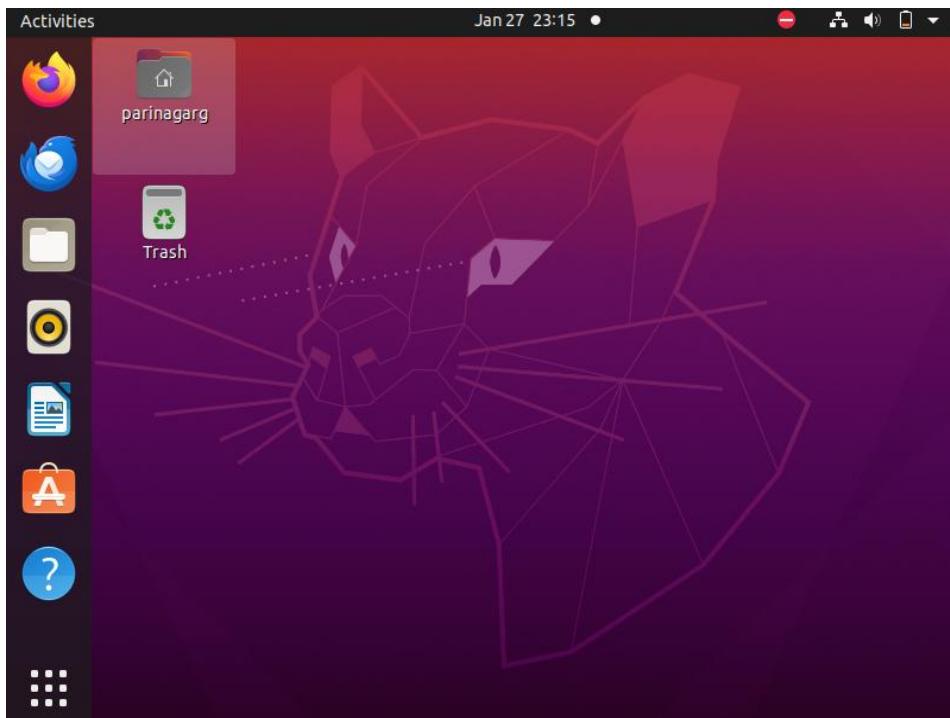
Step 1:



Step 2:



Step 3:

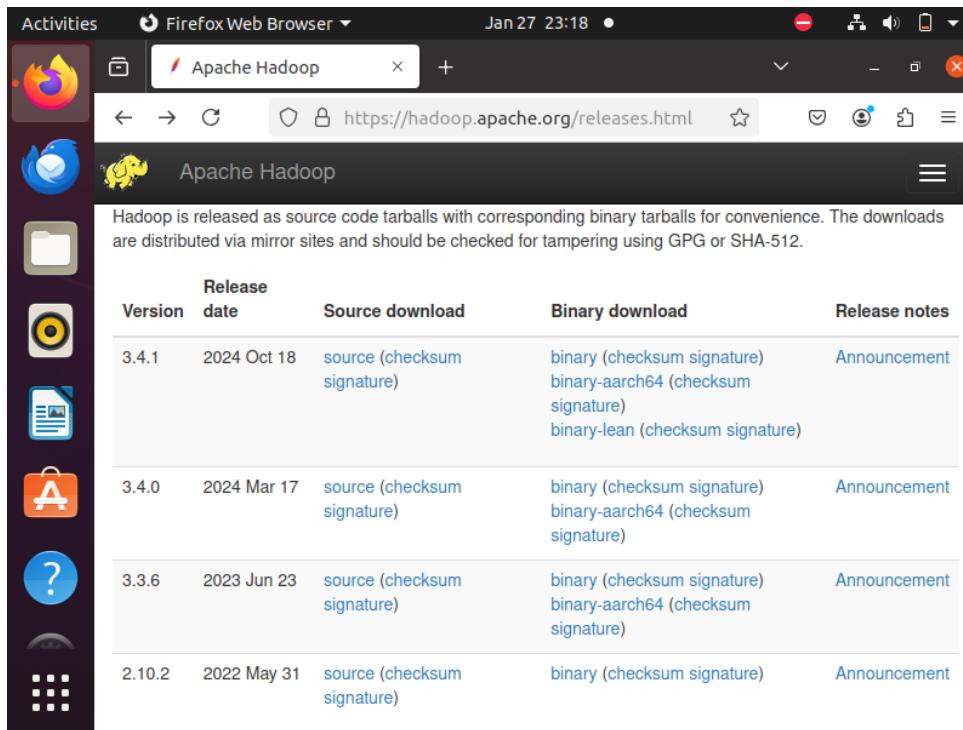


- **Installing Apache Hadoop**

Step 4:

A screenshot of a Firefox browser window. The title bar says "Firefox Web Browser" and the address bar shows the URL "https://hadoop.apache.org". The main content area displays the Apache Hadoop homepage. At the top, there is the Apache Hadoop logo featuring a yellow dog icon and the text "APACHE hadoop". Below the logo, a sub-header reads "Apache Hadoop". The page content explains what Apache Hadoop is: "The Apache® Hadoop® project develops open-source software for reliable, scalable, distributed computing." It then provides a detailed description of the software: "The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures." The browser's sidebar on the left contains icons for various applications, and the top bar includes standard window controls.

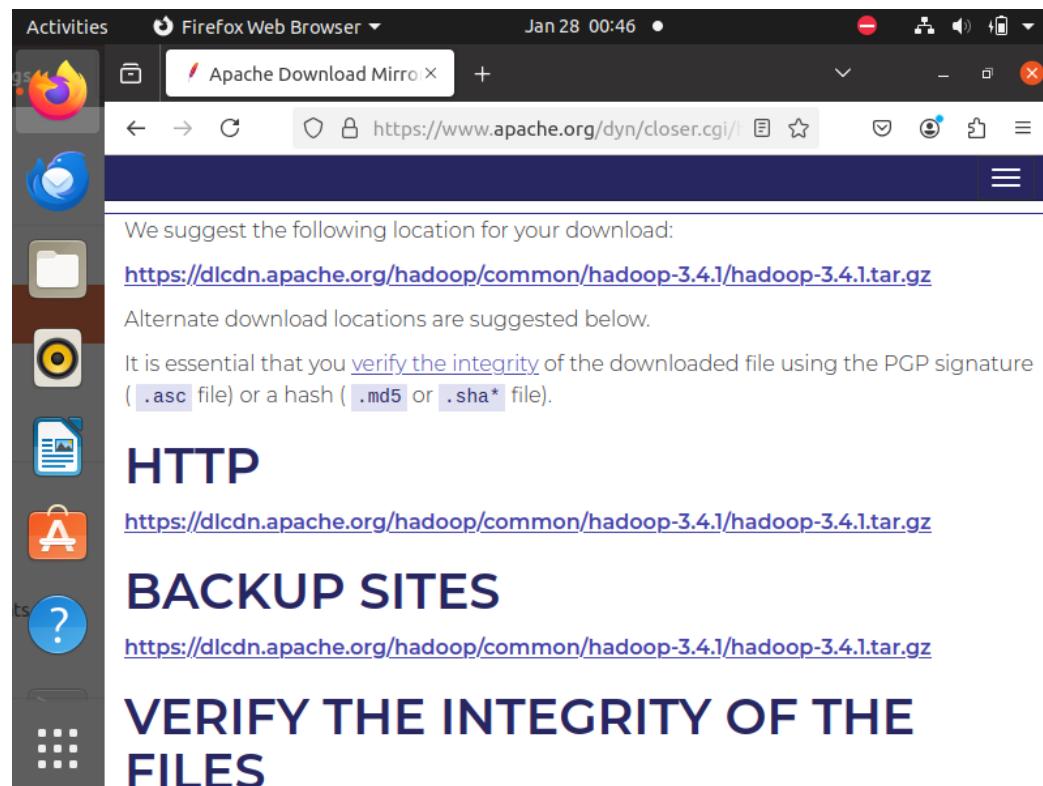
Step 5:



The screenshot shows a Firefox browser window with the title bar "Activities Firefox Web Browser" and the address bar "https://hadoop.apache.org/releases.html". The main content area displays the Apache Hadoop releases page. It features a heading "Hadoop is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-512." Below this is a table titled "Release" with columns: Version, date, Source download, Binary download, and Release notes. The table lists four versions:

Version	date	Source download	Binary download	Release notes
3.4.1	2024 Oct 18	source (checksum signature)	binary (checksum signature) binary-aarch64 (checksum signature) binary-lean (checksum signature)	Announcement
3.4.0	2024 Mar 17	source (checksum signature)	binary (checksum signature) binary-aarch64 (checksum signature)	Announcement
3.3.6	2023 Jun 23	source (checksum signature)	binary (checksum signature) binary-aarch64 (checksum signature)	Announcement
2.10.2	2022 May 31	source (checksum signature)	binary (checksum signature)	Announcement

Step 6:



The screenshot shows a Firefox browser window with the title bar "Activities Firefox Web Browser" and the address bar "https://www.apache.org/dyn/closer.cgi/hadoop-project.cgi?". The main content area displays the Apache Download Mirror page for the Hadoop project. It starts with a message: "We suggest the following location for your download: <https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz>". Below this, it says: "Alternate download locations are suggested below." and provides instructions: "It is essential that you verify the integrity of the downloaded file using the PGP signature ([.asc](#) file) or a hash ([.md5](#) or [.sha*](#) file).". The page then lists download links under three sections: "HTTP", "BACKUP SITES", and "VERIFY THE INTEGRITY OF THE FILES".

We suggest the following location for your download:
<https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz>

Alternate download locations are suggested below.

It is essential that you verify the integrity of the downloaded file using the PGP signature ([.asc](#) file) or a hash ([.md5](#) or [.sha*](#) file).

HTTP

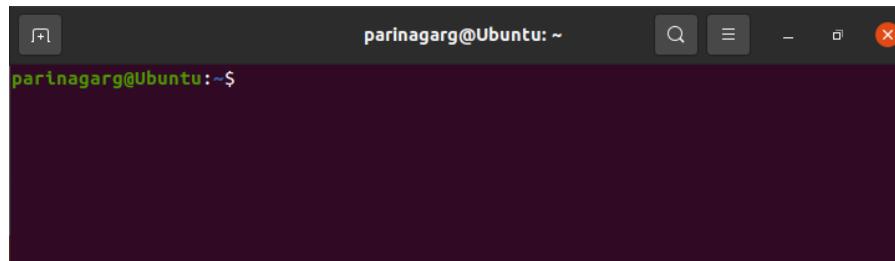
<https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz>

BACKUP SITES

<https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz>

VERIFY THE INTEGRITY OF THE FILES

Step 7:



Step 8:

```
parinagarg@Ubuntu:~$ wget https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz
--2025-01-27 23:24:04-- https://dlcdn.apache.org/hadoop/common/hadoop-3.4.1/hadoop-3.4.1.tar.gz
Resolving dlcdn.apache.org (dlcdn.apache.org)... 151.101.2.132, 2a04:4e42::644
Connecting to dlcdn.apache.org (dlcdn.apache.org)|151.101.2.132|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 974002355 (929M) [application/x-gzip]
Saving to: 'hadoop-3.4.1.tar.gz'

hadoop-3.4.1.tar.gz 100%[=====] 928.88M 4.60MB/s   in 6m 42s
2025-01-27 23:30:48 (2.31 MB/s) - 'hadoop-3.4.1.tar.gz' saved [974002355/974002355]

parinagarg@Ubuntu:~$
```

Step 9:

```
parinagarg@Ubuntu:~$ tar xzf hadoop-3.4.1.tar.gz
parinagarg@Ubuntu:~$
```

- **Installing Java**

Step 10:

```
parinagarg@Ubuntu:~$ sudo apt install default-jre
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java default-jre-headless fonts-dejavu-extra java-common
  libatk-wrapper-java libatk-wrapper-java-jni openjdk-11-jre
  openjdk-11-jre-headless
Suggested packages:
  fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei
  | fonts-wqy-zenhei
The following NEW packages will be installed:
  ca-certificates-java default-jre default-jre-headless fonts-dejavu-extra
  java-common libatk-wrapper-java libatk-wrapper-java-jni openjdk-11-jre
  openjdk-11-jre-headless
0 upgraded, 9 newly installed, 0 to remove and 355 not upgraded.
```

- Setting Up Environment

Step 11:

```
parinagarg@Ubuntu:~$ sudo gedit /etc/profile
```

Step 12:

```
22 export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
23 export PATH=$JAVA_HOME/bin:$PATH
24
25 export HADOOP_HOME=/usr/local/hadoop
26 export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
27 export PATH=$HADOOP_HOME/bin:$HADOOP_HOME/sbin:$PATH
```

Step 13:

```
parinagarg@Ubuntu:~$ source /etc/profile
parinagarg@Ubuntu:~$ echo $JAVA_HOME
/usr/lib/jvm/java-11-openjdk-amd64
```

- Java path and Version

Step 14:

```
parinagarg@Ubuntu:~$ echo $JAVA_HOME
/usr/lib/jvm/java-11-openjdk-amd64
parinagarg@Ubuntu:~$ java -version
openjdk version "11.0.25" 2024-10-15
OpenJDK Runtime Environment (build 11.0.25+9-post-Ubuntu-1ubuntu120.04)
OpenJDK 64-Bit Server VM (build 11.0.25+9-post-Ubuntu-1ubuntu120.04, mixed mode
, sharing)
```

Step 15:

```
parinagarg@Ubuntu:~$ echo $HADOOP_HOME
/usr/local/hadoop
parinagarg@Ubuntu:~$
```

Step 16: _____

```
parinagarg@Ubuntu:~$ hadoop version
Hadoop 3.4.1
Source code repository https://github.com/apache/hadoop.git -r 4d78253093489563
36b8f06a08322b78422849b1
Compiled by mthakur on 2024-10-09T14:57Z
Compiled on platform linux-x86_64
Compiled with protoc 3.23.4
From source with checksum 7292fe9dba5e2e44e3a9f763fce3e680
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-
3.4.1.jar
parinagarg@Ubuntu:~$ █
```

- **Learning Outcomes:**

EXPERIMENT-2

Title :- Develop a map reduce program to calculate the frequency of a given word in a given file.

Introduction -

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.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 WordCount {

    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 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, "word count");
        job.setJarByClass(WordCount.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);
    }
}
```

Step 1:

```
parinagarg@Ubuntu:~$ nano WordCount.java
```

Step 2:

The screenshot shows a terminal window titled "parinagarg@Ubuntu: ~". Inside the terminal, the "nano" text editor is open, displaying the Java code for "WordCount.java". The code imports various Hadoop classes and defines a "WordCount" class with a "TokenizerMapper" inner class. At the bottom of the terminal, there is a status bar with keyboard shortcuts for nano editor commands.

```
GNU nano 4.8          WordCount.java
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 WordCount {

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

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

        [ Wrote 69 lines ]
        ^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
        ^X Exit         ^R Read File     ^\ Replace       ^U Paste Text    ^T To Spell
    }
}
```

Step 3:

```
parinagarg@Ubuntu:~$ javac -classpath $(hadoop classpath) -d . WordCount.java
parinagarg@Ubuntu:~$
```

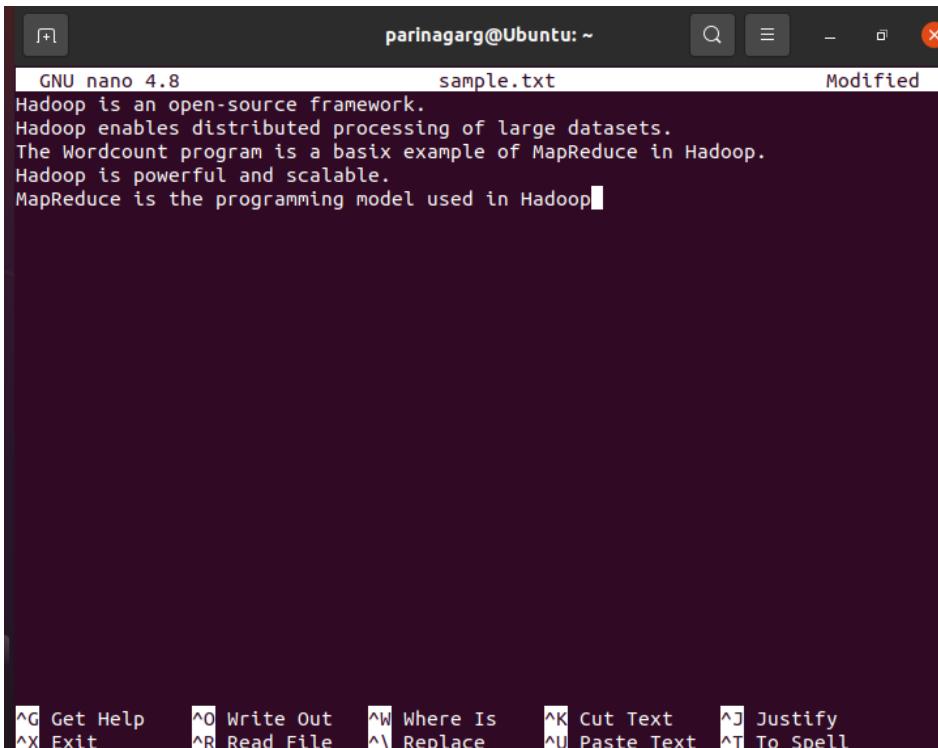
Step 4:

```
parinagarg@Ubuntu:~$ jar cvf wc.jar *.class
added manifest
adding: WordCount$IntSumReducer.class(in = 1755) (out= 750)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 762)(deflated 56%)
adding: WordCount.class(in = 1511) (out= 832)(deflated 44%)
parinagarg@Ubuntu:~$
```

Step 5:

```
parinagarg@Ubuntu:~$ nano sample.txt
parinagarg@Ubuntu:~$
```

Step 6:



The screenshot shows the nano text editor interface. The title bar says "parinagarg@Ubuntu: ~". The file name is "sample.txt" and it is marked as "Modified". The main area contains the following text:

```
GNU nano 4.8          sample.txt          Modified
Hadoop is an open-source framework.
Hadoop enables distributed processing of large datasets.
The Wordcount program is a basic example of MapReduce in Hadoop.
Hadoop is powerful and scalable.
MapReduce is the programming model used in Hadoop.
```

At the bottom, there is a menu of keyboard shortcuts:

```
^G Get Help   ^O Write Out   ^W Where Is   ^K Cut Text   ^J Justify
^X Exit       ^R Read File   ^\ Replace    ^U Paste Text  ^T To Spell
```

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar wc.jar WordCount sample.txt output
2025-02-06 02:25:47,553 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2025-02-06 02:25:47,738 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot
period at 10 second(s).
2025-02-06 02:25:47,741 INFO impl.MetricsSystemImpl: JobTracker metrics system
started
```

Step 8:

```
Map input records=5
Map output records=36
Map output bytes=385
Map output materialized bytes=363
Input split bytes=97
Combine input records=36
Combine output records=27
Reduce input groups=27
Reduce shuffle bytes=363
Reduce input records=27
Reduce output records=27
Spilled Records=54
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=18
Total committed heap usage (bytes)=406847488
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=241
File Output Format Counters
Bytes Written=261
parinagarg@Ubuntu:~$
```

Step 9:

```
parinagarg@Ubuntu:~$ hdfs dfs -cat output/part-r-00000
Hadoop 4
Hadoop. 1
MapReduce 2
The 1
Wordcount 1
a 1
an 1
and 1
basix 1
datasets. 1
distributed 1
enables 1
example 1
framework. 1
in 2
is 4
large 1
model 1
of 2
open-source 1
powerful 1
processing 1
program 1
programming 1
scalable. 1
the 1
used 1
parinagarg@Ubuntu:~$
```

Learning Outcomes:

EXPERIMENT-3

Title :- Develop a map reduce program to find the maximum temperature in each year.

Introduction -

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.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 Max_temp {
    public static class TemperatureMapper
        extends Mapper<Object, Text, Text, IntWritable>{

        private Text year = new Text();
        private IntWritable temperature = new IntWritable();

        public void map(Object key, Text value, Context context
                        ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());

            if (itr.hasMoreTokens()) {
                String yearString = itr.nextToken();
                String temperatureString = itr.nextToken();

                try {
                    year.set(yearString);
                    temperature.set(Integer.parseInt(temperatureString));
                    context.write(year, temperature);
                } catch (NumberFormatException e) {

                    System.err.println("Skipping invalid record: " +
                        value.toString());
                }
            }
        }
    }

    public static class MaxTemperatureReducer
        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 maxTemp = Integer.MIN_VALUE;
            for (IntWritable val : values) {
                maxTemp = Math.max(maxTemp, val.get());
            }
            result.set(maxTemp);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
    }
}
```

```
Job job = Job.getInstance(conf, "Maximum Temperature");

job.setJarByClass(Max_temp.class);

job.setMapperClass(TemperatureMapper.class);
job.setReducerClass(MaxTemperatureReducer.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);
}

}
```

Step 1: _____

```
parinagarg@Ubuntu:~$ nano Max_temp.java
parinagarg@Ubuntu:~$
```

Step 2: _____

The screenshot shows a terminal window titled "parinagarg@Ubuntu: ~". The command "nano Max_temp.java" is run, creating a new file named "max_temp.java". The code is a Java class named "Max_temp" that extends "Mapper<Object, Text, Text, IntWritable>". It includes imports for various Hadoop classes and defines a "map" method that tokenizes input and emits temperature values.

```
GNU nano 4.8                         max_temp.java                         Modified
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 Max_temp {
    public static class TemperatureMapper
        extends Mapper<Object, Text, Text, IntWritable> {

        private Text year = new Text();
        private IntWritable temperature = new IntWritable();

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

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

Step 3: _____

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) Max_temp.java
parinagarg@Ubuntu:~$
```

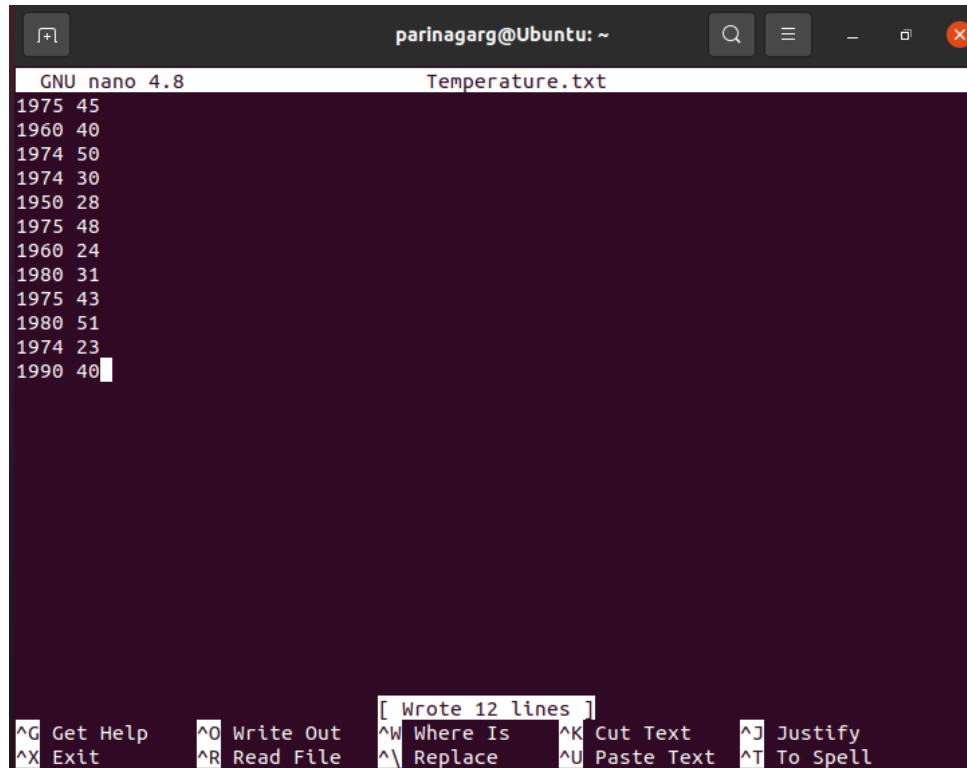
Step 4: _____

```
parinagarg@Ubuntu:~$ jar cvf mt.jar *.class
added manifest
adding: Max_temp$MaxTemperatureReducer.class(in = 1843) (out= 796)(deflated 56%)
adding: Max_temp$TemperatureMapper.class(in = 2533) (out= 1114)(deflated 56%)
adding: Max_temp.class(in = 1497) (out= 810)(deflated 45%)
adding: WordCount$IntSumReducer.class(in = 1755) (out= 750)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 762)(deflated 56%)
adding: WordCount.class(in = 1511) (out= 832)(deflated 44%)
parinagarg@Ubuntu:~$
```

Step 5:

```
parinagarg@Ubuntu:~$ nano Temperature.txt  
parinagarg@Ubuntu:~$
```

Step 6:



The screenshot shows the nano text editor window on a dark-themed desktop environment. The title bar reads "parinagarg@Ubuntu: ~" and the file name is "Temperature.txt". The editor displays the following text:

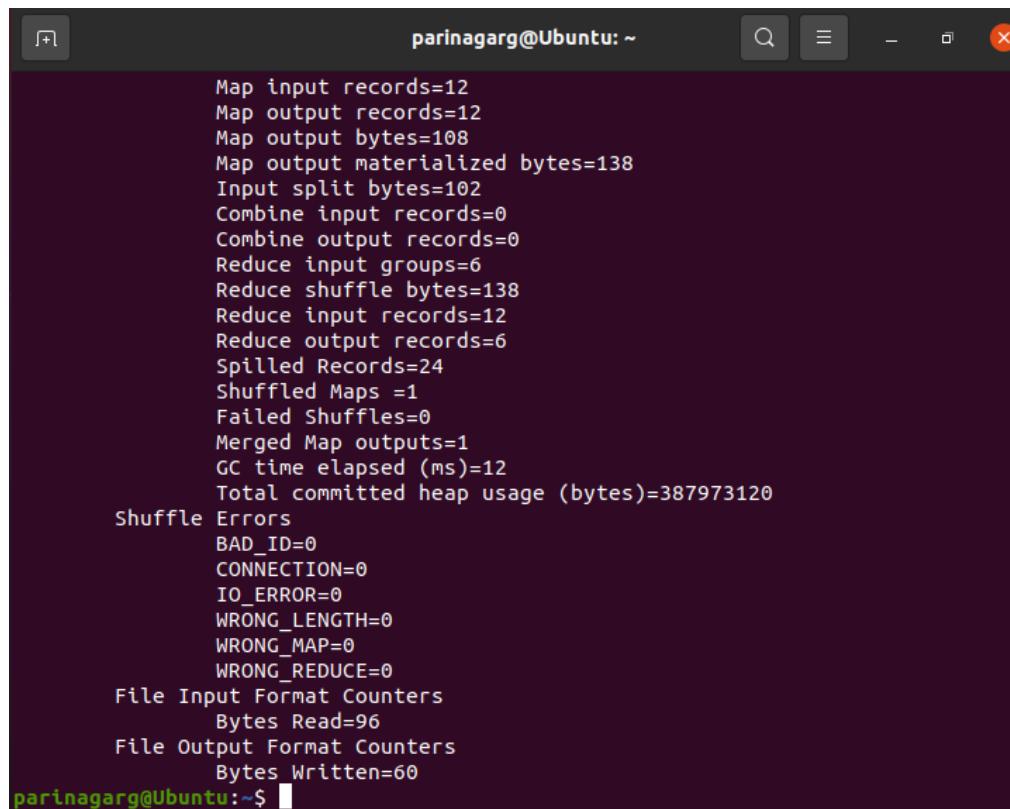
```
GNU nano 4.8  
1975 45  
1960 40  
1974 50  
1974 30  
1950 28  
1975 48  
1960 24  
1980 31  
1975 43  
1980 51  
1974 23  
1990 40
```

At the bottom of the editor, there is a status bar with the message "[Wrote 12 lines]". Below the status bar, a menu bar lists various keyboard shortcuts for editing.

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar mt.jar Max_temp Temperature.txt output_2  
2025-02-09 22:29:51,676 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties  
2025-02-09 22:29:51,873 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).  
2025-02-09 22:29:51,873 INFO impl.MetricsSystemImpl: JobTracker metrics system started  
2025-02-09 22:29:52,008 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
2025-02-09 22:29:52,100 INFO input.FileInputFormat: Total input files to process : 1
```

Step 8:

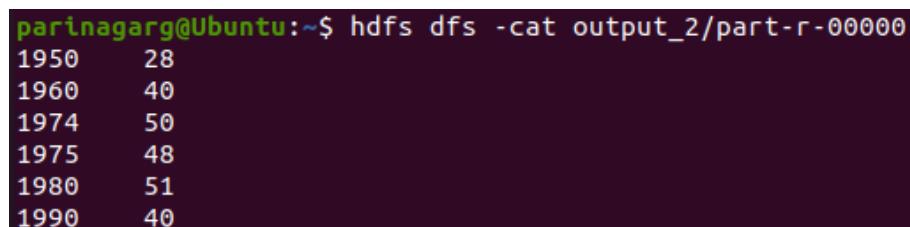


A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The window displays various Hadoop job metrics. The output includes:

```
Map input records=12
Map output records=12
Map output bytes=108
Map output materialized bytes=138
Input split bytes=102
Combine input records=0
Combine output records=0
Reduce input groups=6
Reduce shuffle bytes=138
Reduce input records=12
Reduce output records=6
Spilled Records=24
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=12
Total committed heap usage (bytes)=387973120
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=96
File Output Format Counters
Bytes Written=60
```

parinagarg@Ubuntu:~\$

Step 9:



A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The command "hdfs dfs -cat output_2/part-r-00000" is run, displaying the following data:

```
1950    28
1960    40
1974    50
1975    48
1980    51
1990    40
```

Learning Outcomes:

EXPERIMENT-4

Title :- Develop a map reduce program to find the grade of students.

Introduction -

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.Text;
import org.apache.hadoop.io.IntWritable;
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 StudentGrades {

    public static class GradeMapper extends Mapper<Object, Text, Text, Text> {
        private Text studentName = new Text();
        private Text grade = new Text();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());

            if (itr.hasMoreTokens()) {
                String name = itr.nextToken(); // Student name
                String marksStr = itr.nextToken(); // Marks

                try {
                    int marks = Integer.parseInt(marksStr);
                    String gradeLetter;

                    if (marks >= 90) {
                        gradeLetter = "A";
                    } else if (marks >= 80) {
                        gradeLetter = "B";
                    } else if (marks >= 70) {
                        gradeLetter = "C";
                    } else if (marks >= 60) {
                        gradeLetter = "D";
                    } else {
                        gradeLetter = "F";
                    }

                    studentName.set(name);
                    grade.set(gradeLetter);
                    context.write(studentName, grade);

                } catch (NumberFormatException e) {
                    System.err.println("Skipping invalid record: " + value.toString());
                }
            }
        }
    }

    public static class GradeReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            for (Text val : values) {
                context.write(key, val); // Output (StudentName -> Grade)
            }
        }
    }
}
```

```
        }
    }

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Student Grades Calculation");

    job.setJarByClass(StudentGrades.class);
    job.setMapperClass(GradeMapper.class);
    job.setReducerClass(GradeReducer.class);

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

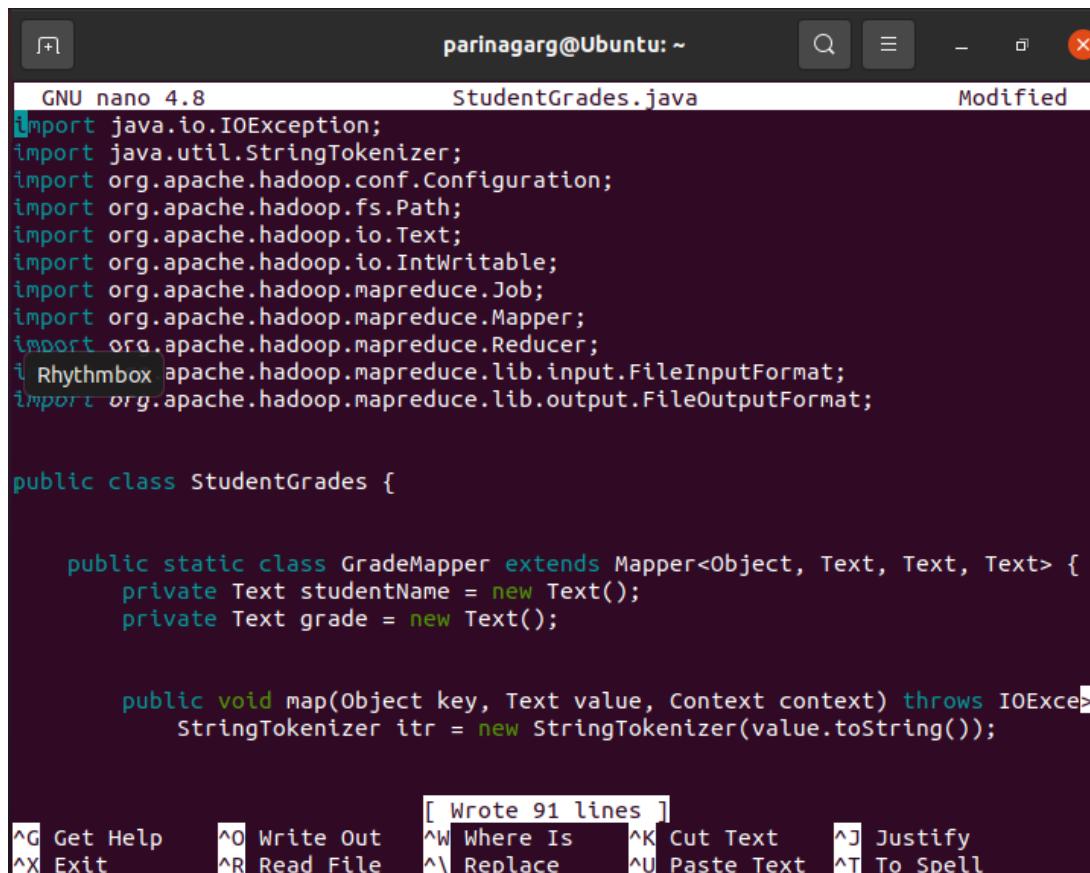
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

Step 1: _____

```
parinagarg@Ubuntu:~$ nano StudentGrades.java
parinagarg@Ubuntu:~$ █
```

Step 2: _____



The screenshot shows the nano text editor window on a Linux desktop. The title bar says "parinagarg@Ubuntu: ~". The status bar at the bottom indicates "[Wrote 91 lines]". The main area contains Java code for a MapReduce job. The code includes imports for IOException, StringTokenizer, Configuration, Path, Text, IntWritable, Job, Mapper, Reducer, FileInputFormat, and FileOutputFormat. It defines a public class StudentGrades with a static inner class GradeMapper that extends Mapper<Object, Text, Text, Text>. The map method tokenizes the input value. At the bottom of the editor, there are various keyboard shortcut keys listed.

```
GNU nano 4.8                               StudentGrades.java                               Modified
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.Text;
import org.apache.hadoop.io.IntWritable;
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 StudentGrades {

    public static class GradeMapper extends Mapper<Object, Text, Text, Text> {
        private Text studentName = new Text();
        private Text grade = new Text();

        public void map(Object key, Text value, Context context) throws IOException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            studentName.set(itr.nextToken());
            grade.set(itr.nextToken());
            context.write(studentName, grade);
        }

        protected void cleanup(Context context) throws IOException {
            super.cleanup(context);
            context.write(studentName, grade);
        }
    }

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

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

    public static class WordCountMapper extends Mapper<Text, Text, Text, IntWritable> {
        private IntWritable one = new IntWritable(1);

        public void map(Text key, Text value, Context context) throws IOException {
            context.write(key, one);
        }
    }

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

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

[Wrote 91 lines]

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

Step 3: _____

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) StudentGrades.java
parinagarg@Ubuntu:~$
```

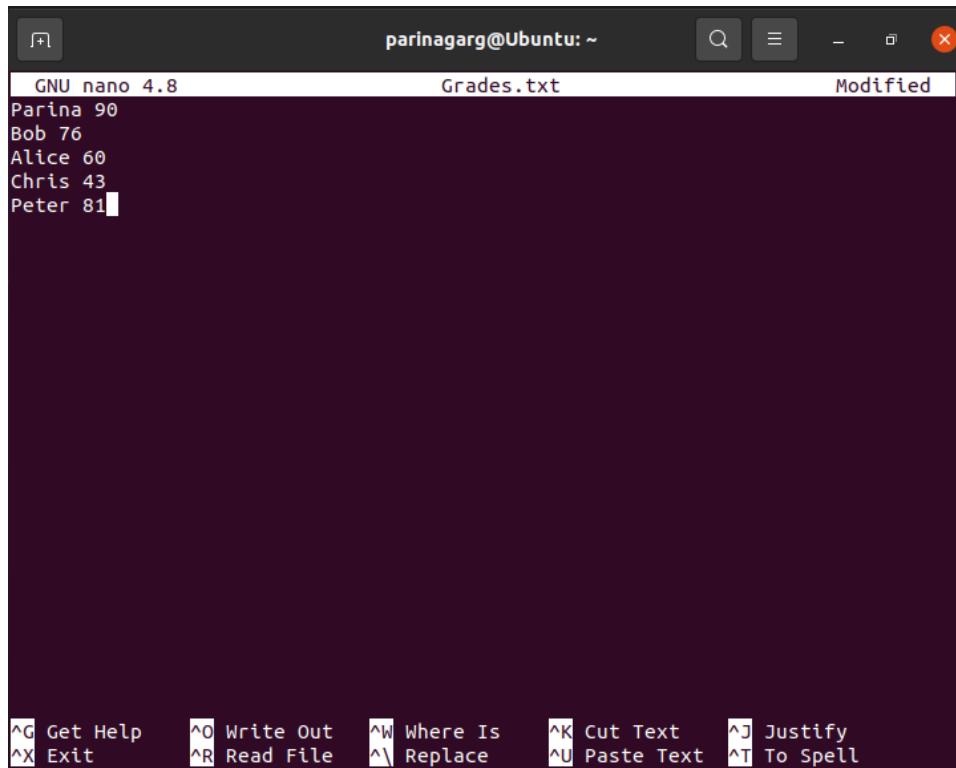
Step 4: _____

```
parinagarg@Ubuntu:~$ jar cvf sg.jar *.class
added manifest
adding: Max_temp$MaxTemperatureReducer.class(in = 1843) (out= 796)(deflated 56%)
adding: Max_temp$TemperatureMapper.class(in = 2533) (out= 1114)(deflated 56%)
adding: Max_temp.class(in = 1497) (out= 810)(deflated 45%)
adding: StudentGrades$GradeMapper.class(in = 2595) (out= 1185)(deflated 54%)
adding: StudentGrades$GradeReducer.class(in = 1526) (out= 635)(deflated 58%)
adding: StudentGrades.class(in = 1456) (out= 793)(deflated 45%)
adding: WordCount$IntSumReducer.class(in = 1755) (out= 750)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 762)(deflated 56%)
adding: WordCount.class(in = 1511) (out= 832)(deflated 44%)
```

Step 5:

```
parinagarg@Ubuntu:~$ nano Grades.txt  
parinagarg@Ubuntu:~$
```

Step 6:



The screenshot shows the nano text editor window. The title bar says "parinagarg@Ubuntu: ~". The file name "Grades.txt" is in the center, and "Modified" is in the status bar. The main area contains the following text:

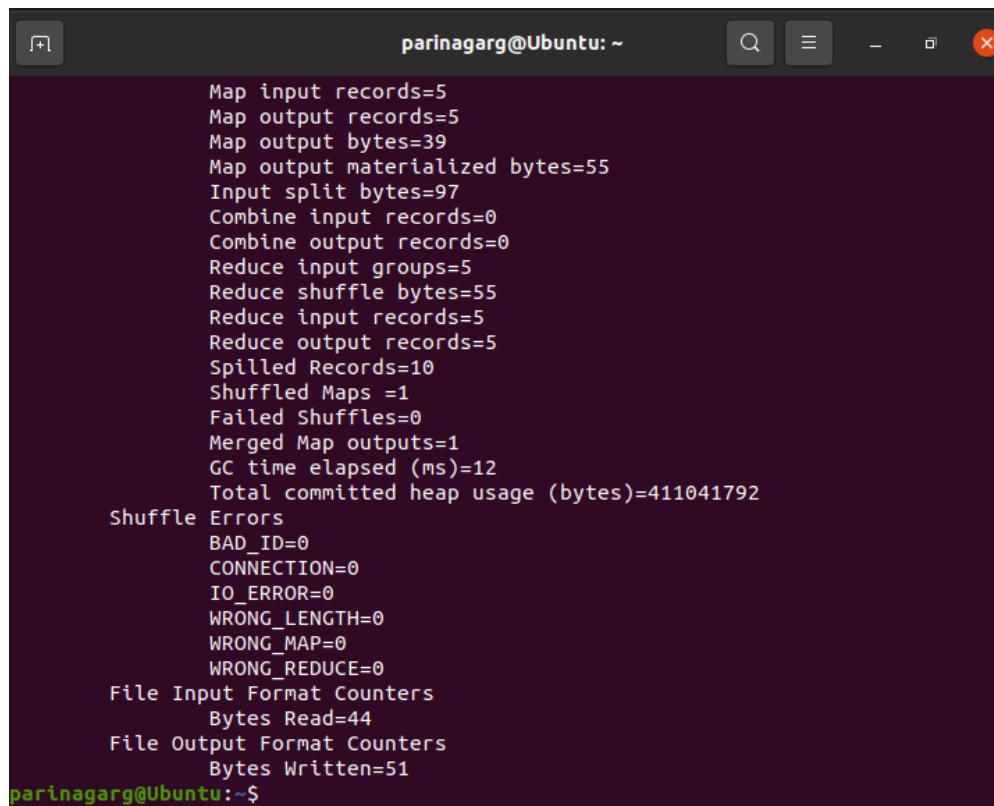
```
GNU nano 4.8  
Parina 90  
Bob 76  
Alice 60  
Chris 43  
Peter 81
```

At the bottom, there is a menu bar with options: Get Help, Write Out, Where Is, Cut Text, Justify, Exit, Read File, Replace, Paste Text, To Spell.

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar sg.jar StudentGrades Grades.txt output_3  
2025-02-09 22:44:18,806 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties  
2025-02-09 22:44:18,909 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).  
2025-02-09 22:44:18,910 INFO impl.MetricsSystemImpl: JobTracker metrics system started  
2025-02-09 22:44:18,970 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
2025-02-09 22:44:19,036 INFO input.FileInputFormat: Total input files to process : 1
```

Step 8:

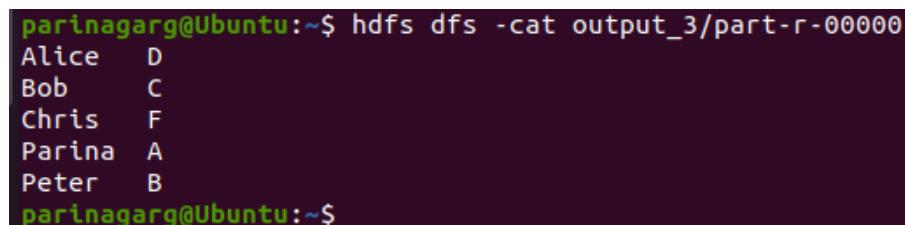


A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The window displays various Hadoop job metrics. The output includes:

```
Map input records=5
Map output records=5
Map output bytes=39
Map output materialized bytes=55
Input split bytes=97
Combine input records=0
Combine output records=0
Reduce input groups=5
Reduce shuffle bytes=55
Reduce input records=5
Reduce output records=5
Spilled Records=10
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=12
Total committed heap usage (bytes)=411041792
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=44
File Output Format Counters
Bytes Written=51
```

parinagarg@Ubuntu:~\$

Step 9:



A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The command "hdfs dfs -cat output_3/part-r-00000" is run, displaying the following data:

```
Alice D
Bob C
Chris F
Parina A
Peter B
```

parinagarg@Ubuntu:~\$

Learning Outcomes:

EXPERIMENT-5

Title :- Develop a map reduce program to implement matrix multiplication.

Introduction -

Code:

```
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;

import java.io.IOException;
import java.util.ArrayList;

public class MatrixMultiplication {

    public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String[] parts = value.toString().split(" ");
            String matrixName = parts[0];
            int i = Integer.parseInt(parts[1]);
            int j = Integer.parseInt(parts[2]);
            int val = Integer.parseInt(parts[3]);

            if (matrixName.equals("A")) {
                for (int k = 0; k < 2; k++) {
                    context.write(new Text(i + "," + k), new Text("A," + j + "," + val));
                }
            } else {
                for (int k = 0; k < 2; k++) { // 
                    context.write(new Text(k + "," + j), new Text("B," + i + "," + val));
                }
            }
        }
    }

    public static class MatrixReducer extends Reducer<Text, Text, Text, IntWritable> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            ArrayList<String> A = new ArrayList<>();
            ArrayList<String> B = new ArrayList<>();

            for (Text val : values) {
                String value = val.toString();
                if (value.startsWith("A")) {
                    A.add(value.substring(2));
                } else {
                    B.add(value.substring(2));
                }
            }
        }
    }
}
```

```

        int sum = 0;
        for (String a : A) {
            for (String b : B) {
                String[] aParts = a.split(",");
                String[] bParts = b.split(",");
                if (aParts[0].equals(bParts[0])) {
                    sum += Integer.parseInt(aParts[1]) * Integer.parseInt(bParts[1]);
                }
            }
        }
        context.write(key, new IntWritable(sum));
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Matrix Multiplication");
    job.setJarByClass(MatrixMultiplication.class);
    job.setMapperClass(MatrixMapper.class);
    job.setReducerClass(MatrixReducer.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.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);
}
}

```

Step 1:

```
parinagarg@Ubuntu:~$ nano MatrixMultiplication.java
parinagarg@Ubuntu:~$
```

Step 2:

The screenshot shows a terminal window titled "parinagarg@Ubuntu: ~". Inside the terminal, the "nano" text editor is open, displaying the Java code for "MatrixMultiplication". The code includes imports for various Hadoop classes and a main class definition. At the bottom of the terminal, there is a status bar with various keyboard shortcuts.

```
GNU nano 4.8          MatrixMultiplication.java
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;

import java.io.IOException;
import java.util.ArrayList;

public class MatrixMultiplication {

    public static class MatrixMapper extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context) throws IOException {
            String[] parts = value.toString().split(" ");
            String matrixName = parts[0];
            int i = Integer.parseInt(parts[1]);
            int j = Integer.parseInt(parts[2]);
            int val = Integer.parseInt(parts[3]);

            if (matrixName.equals("A")) {
                for (int k = 0; k < 2; k++) { // Adjust size as needed
                    [ Read 78 lines ]

```

AG Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
AX Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) MatrixMultiplication.java
parinagarg@Ubuntu:~$
```

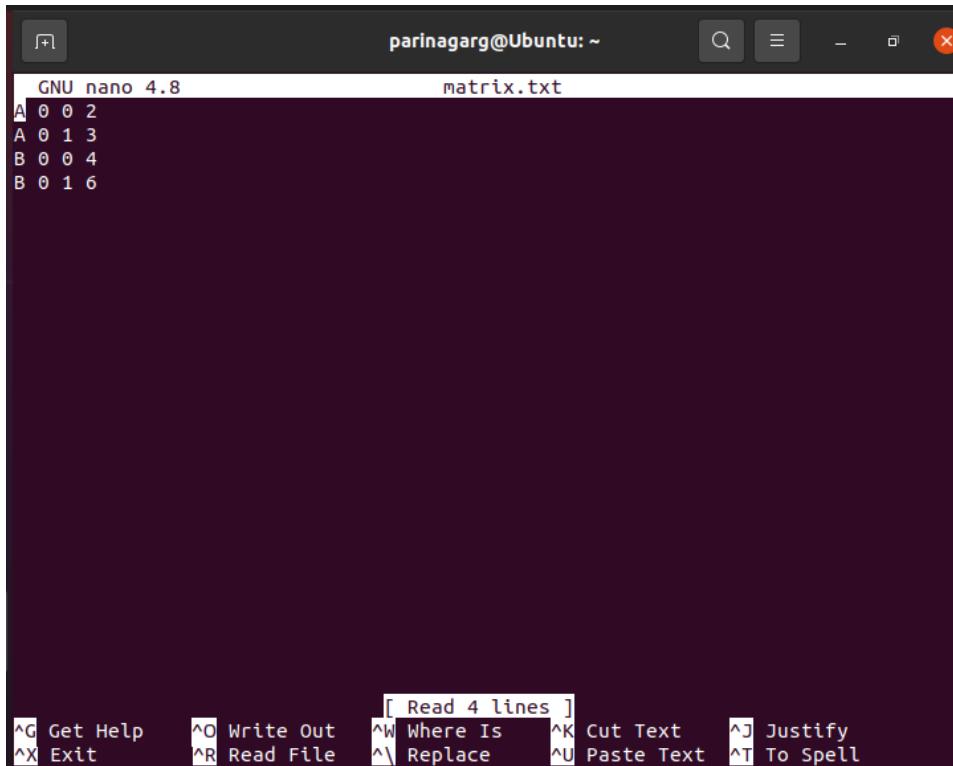
Step 4:

```
parinagarg@Ubuntu:~$ jar cvf mm.jar *.class
added manifest
adding: MatrixMultiplication$MatrixMapper.class(in = 2297) (out= 989)(deflated 56%)
adding: MatrixMultiplication$MatrixReducer.class(in = 2304) (out= 1052)(deflated 54%)
adding: MatrixMultiplication.class(in = 1609) (out= 842)(deflated 47%)
adding: Max_temp$MaxTemperatureReducer.class(in = 1843) (out= 796)(deflated 56%)
adding: Max_temp$TemperatureMapper.class(in = 2533) (out= 1114)(deflated 56%)
adding: Max_temp.class(in = 1497) (out= 810)(deflated 45%)
adding: StudentGrades$GradeMapper.class(in = 2595) (out= 1185)(deflated 54%)
adding: StudentGrades$GradeReducer.class(in = 1526) (out= 635)(deflated 58%)
adding: StudentGrades.class(in = 1456) (out= 793)(deflated 45%)
adding: WordCount$IntSumReducer.class(in = 1755) (out= 750)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 762)(deflated 56%)
adding: WordCount.class(in = 1511) (out= 832)(deflated 44%)
```

Step 5:

```
parinagarg@Ubuntu:~$ nano matrix.txt
parinagarg@Ubuntu:~$
```

Step 6:



A screenshot of the nano text editor window. The title bar says "parinagarg@Ubuntu: ~". The file name "matrix.txt" is shown in the title bar. The main area contains the following text:

```
GNU nano 4.8
matrix.txt
A 0 0 2
A 0 1 3
B 0 0 4
B 0 1 6
```

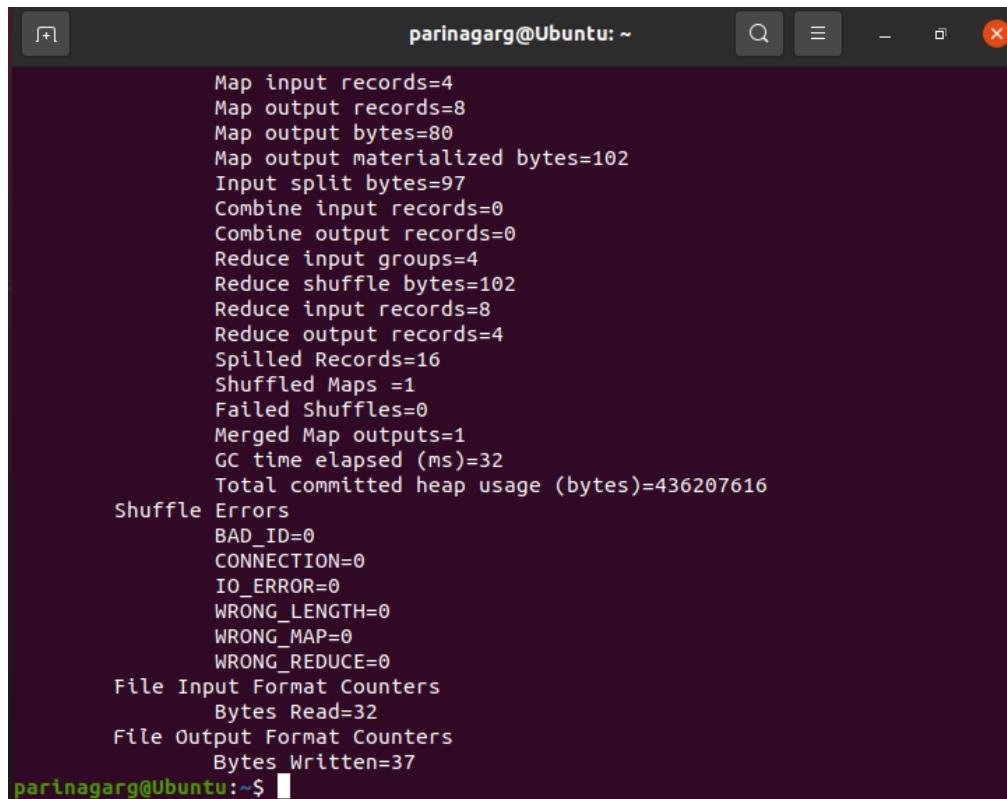
The bottom status bar shows keyboard shortcuts:

```
[ Read 4 lines ]
^G Get Help    ^O Write Out    [ ] Where Is    ^K Cut Text    ^J Justify
^X Exit        ^R Read File    ^\ Replace     ^U Paste Text   ^T To Spell
```

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar mm.jar MatrixMultiplication matrix.txt output_1
2025-02-11 10:26:17,505 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2025-02-11 10:26:17,657 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2025-02-11 10:26:17,657 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2025-02-11 10:26:17,761 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2025-02-11 10:26:17,926 INFO input.FileInputFormat: Total input files to process : 1
```

Step 8:

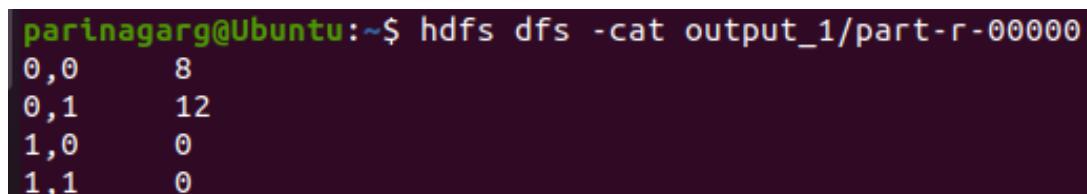


A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The window displays various Hadoop job metrics. The text output includes:

```
Map input records=4
Map output records=8
Map output bytes=80
Map output materialized bytes=102
Input split bytes=97
Combine input records=0
Combine output records=0
Reduce input groups=4
Reduce shuffle bytes=102
Reduce input records=8
Reduce output records=4
Spilled Records=16
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=32
Total committed heap usage (bytes)=436207616
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=32
File Output Format Counters
Bytes Written=37
```

The prompt at the bottom is "parinagarg@Ubuntu:~\$".

Step 9:



A screenshot of a terminal window titled "parinagarg@Ubuntu: ~\$". The command "hdfs dfs -cat output_1/part-r-00000" is run, and the output shows four lines of data:

```
0,0      8
0,1      12
1,0      0
1,1      0
```

Learning Outcomes:

EXPERIMENT-6

Title :- Develop a map reduce program to find the maximum electrical consumption in each year given electrical consumption for each month in each year.

Introduction -

Code:

```
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;
import java.io.IOException;

public class MaxElectricConsumption {
    public static class ConsumptionMapper extends Mapper<Object, Text, Text, IntWritable> {
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] fields = value.toString().split("\\s+");
            if (fields.length == 3) {
                String year = fields[0];
                int consumption = Integer.parseInt(fields[2]);
                context.write(new Text(year), new IntWritable(consumption));
            }
        }
    }

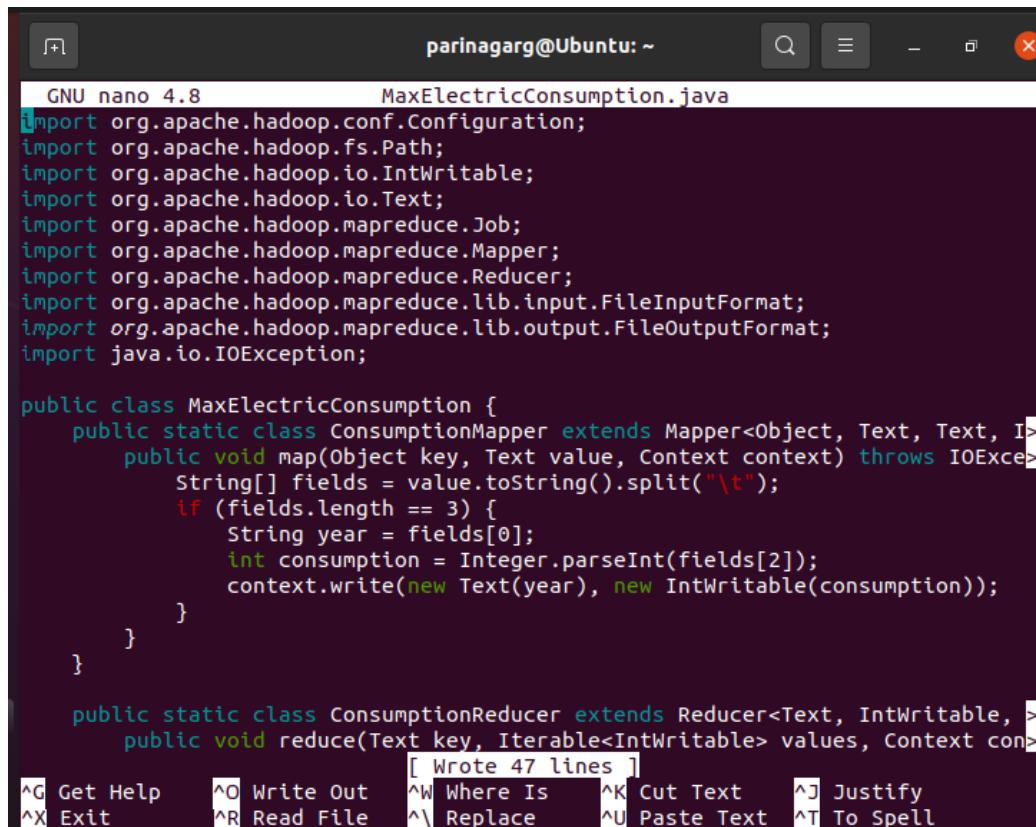
    public static class ConsumptionReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            int maxConsumption = Integer.MIN_VALUE;
            for (IntWritable val : values) {
                maxConsumption = Math.max(maxConsumption, val.get());
            }
            context.write(key, new IntWritable(maxConsumption));
        }
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "max electrical consumption");
    job.setJarByClass(MaxElectricConsumption.class);
    job.setMapperClass(ConsumptionMapper.class);
    job.setCombinerClass(ConsumptionReducer.class);
    job.setReducerClass(ConsumptionReducer.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);
}
```

Step 1:

```
parinagarg@Ubuntu:~$ nano MaxElectricConsumption.java
parinagarg@Ubuntu:~$
```

Step 2:



```
GNU nano 4.8          MaxElectricConsumption.java
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;
import java.io.IOException;

public class MaxElectricConsumption {
    public static class ConsumptionMapper extends Mapper<Object, Text, Text, IntWritable> {
        public void map(Object key, Text value, Context context) throws IOException {
            String[] fields = value.toString().split("\t");
            if (fields.length == 3) {
                String year = fields[0];
                int consumption = Integer.parseInt(fields[2]);
                context.write(new Text(year), new IntWritable(consumption));
            }
        }
    }

    public static class ConsumptionReducer extends Reducer<Text, IntWritable, > {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException {
            int maxConsumption = 0;
            for (IntWritable value : values) {
                if (value.get() > maxConsumption) {
                    maxConsumption = value.get();
                }
            }
            context.write(key, new IntWritable(maxConsumption));
        }
    }
}

[ Wrote 47 lines ]
^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
^X Exit         ^R Read File     ^\ Replace       ^U Paste Text    ^T To Spell
```

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) -d . MaxElectricConsumption.java
parinagarg@Ubuntu:~$
```

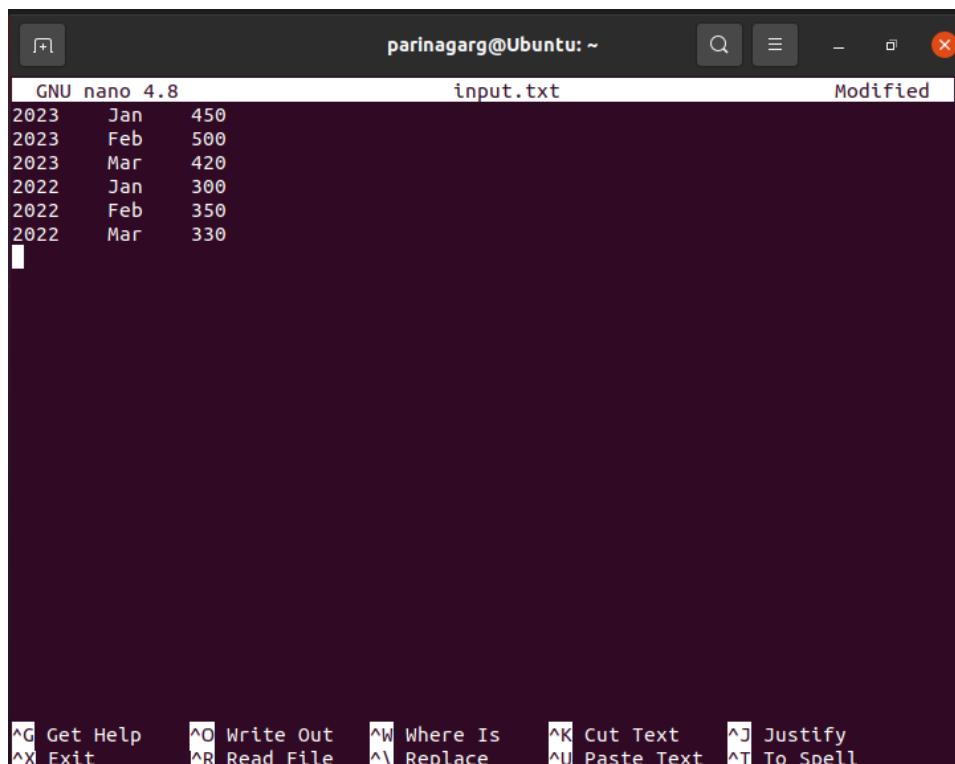
Step 4:

```
parinagarg@Ubuntu:~$ jar cvf mec.jar *.class
added manifest
adding: matrixMultiplication$MatrixMapper.class(in = 2297) (out= 990)(deflated
56%)
adding: MatrixMultiplication$MatrixMapper.class(in = 2297) (out= 989)(deflated
56%)
adding: matrixMultiplication$MatrixReducer.class(in = 2304) (out= 1053)(deflate
d 54%)
adding: MatrixMultiplication$MatrixReducer.class(in = 2304) (out= 1052)(deflate
d 54%)
adding: matrixMultiplication.class(in = 1609) (out= 843)(deflated 47%)
adding: MatrixMultiplication.class(in = 1609) (out= 842)(deflated 47%)
adding: MaxElectricConsumption$ConsumptionMapper.class(in = 1690) (out= 694)(de
flated 58%)
adding: MaxElectricConsumption$ConsumptionReducer.class(in = 1781) (out= 752)(d
eflated 57%)
adding: MaxElectricConsumption.class(in = 1593) (out= 837)(deflated 47%)
adding: Max_temp$MaxTemperatureReducer.class(in = 1843) (out= 796)(deflated 56%
)
adding: Max_temp$TemperatureMapper.class(in = 2533) (out= 1114)(deflated 56%)
adding: Max_temp.class(in = 1497) (out= 810)(deflated 45%)
adding: StudentGrades$GradeMapper.class(in = 2595) (out= 1185)(deflated 54%)
adding: StudentGrades$GradeReducer.class(in = 1526) (out= 635)(deflated 58%)
adding: StudentGrades.class(in = 1456) (out= 793)(deflated 45%)
adding: WordCount$IntSumReducer.class(in = 1755) (out= 750)(deflated 57%)
adding: WordCount$TokenizerMapper.class(in = 1752) (out= 762)(deflated 56%)
adding: WordCount.class(in = 1511) (out= 832)(deflated 44%)
```

Step 5:

```
parinagarg@Ubuntu:~$ nano input.txt
```

Step 6:



The screenshot shows a terminal window with the title bar "parinagarg@Ubuntu: ~". The window contains a nano text editor interface. The status bar at the top shows "GNU nano 4.8", the file name "input.txt", and the status "Modified". The main area of the editor displays the following text:

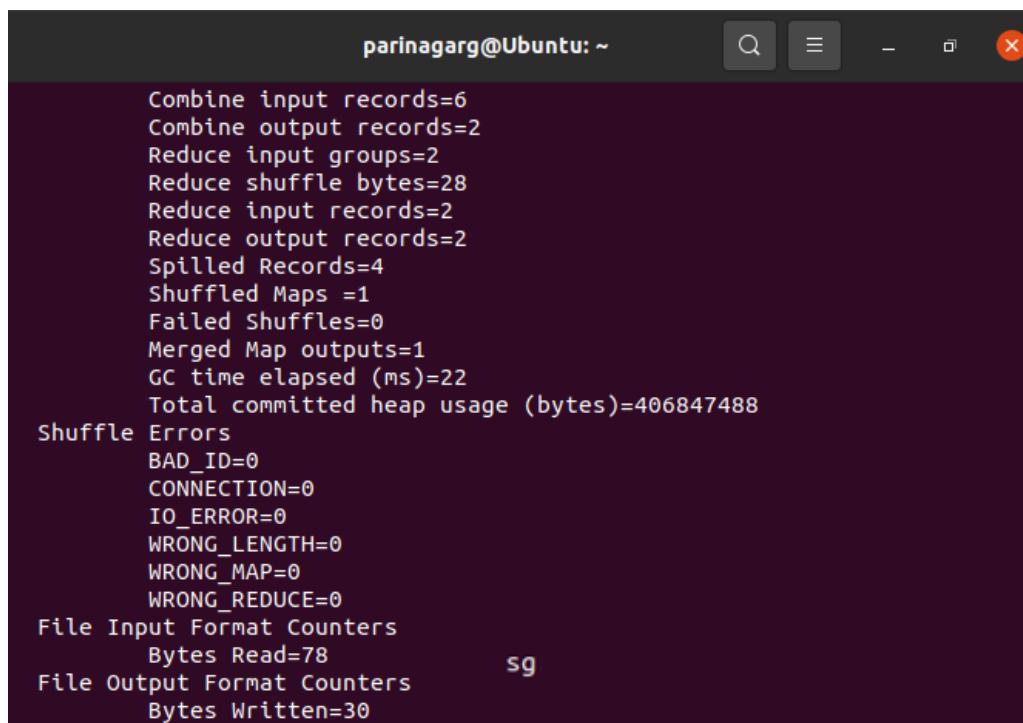
```
2023 Jan 450
2023 Feb 500
2023 Mar 420
2022 Jan 300
2022 Feb 350
2022 Mar 330
```

At the bottom of the editor, there is a menu bar with the following options: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^X Exit, ^R Read File, ^I Replace, ^U Paste Text, and ^T To Spell.

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar mec.jar MaxElectricConsumption input.txt output
_6
2025-02-21 01:25:54,167 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2025-02-21 01:25:54,251 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot
period at 10 second(s).
2025-02-21 01:25:54,251 INFO impl.MetricsSystemImpl: JobTracker metrics system
started
2025-02-21 01:25:54,311 WARN mapreduce.JobResourceUploader: Hadoop command-line
option parsing not performed. Implement the Tool interface and execute your ap-
plication with ToolRunner to remedy this.
2025-02-21 01:25:54,407 INFO input.FileInputFormat: Total input files to proces
s : 1
```

Step 8:



A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The window displays various Hadoop job metrics. The metrics include:

- Combine input records=6
- Combine output records=2
- Reduce input groups=2
- Reduce shuffle bytes=28
- Reduce input records=2
- Reduce output records=2
- Spilled Records=4
- Shuffled Maps =1
- Failed Shuffles=0
- Merged Map outputs=1
- GC time elapsed (ms)=22
- Total committed heap usage (bytes)=406847488
- shuffle Errors
 - BAD_ID=0
 - CONNECTION=0
 - IO_ERROR=0
 - WRONG_LENGTH=0
 - WRONG_MAP=0
 - WRONG_REDUCE=0
- File Input Format Counters
 - Bytes Read=78
- File Output Format Counters
 - Bytes Written=30

Step 9:

```
parinagarg@Ubuntu:~$ hdfs dfs -cat output_6/part-r-00000
2022    350
2023    500
```

Learning Outcomes:

EXPERIMENT-7

Title :- Develop a map reduce program to analyse weather data set and print whether the day is shiny or cool day.

Introduction -

Code:

```
import java.io.IOException;
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 WeatherAnalysis {
    public static class WeatherMapper extends Mapper<Object, Text, Text, IntWritable> {
        private Text date = new Text();
        private IntWritable temperature = new IntWritable();

        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] fields = value.toString().split(",");
            if (fields.length > 1) {
                date.set(fields[0]);
                int temp = Integer.parseInt(fields[1]);
                temperature.set(temp);
                context.write(date, temperature);
            }
        }
    }

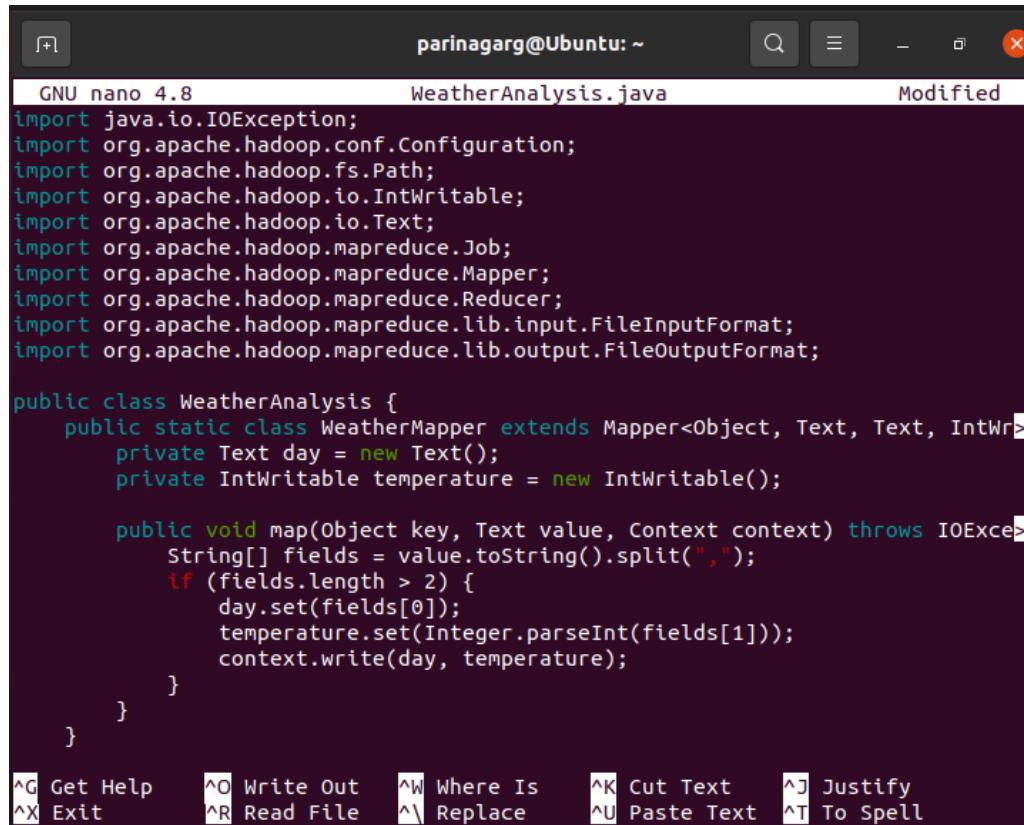
    public static class WeatherReducer extends Reducer<Text, IntWritable, Text, Text> {
        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {
            for (IntWritable val : values) {
                String result = val.get() > 30 ? "Shiny Day" : "Cool Day";
                context.write(key, new Text(result));
            }
        }
    }
}

public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "weather analysis");
    job.setJarByClass(WeatherAnalysis.class);
    job.setMapperClass(WeatherMapper.class);
    job.setReducerClass(WeatherReducer.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);
}
```

Step 1:

```
parinagarg@Ubuntu:~$ nano WeatherAnalysis.java
parinagarg@Ubuntu:~$
```

Step 2:



A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The window shows the "WeatherAnalysis.java" file being edited in the nano text editor. The code defines a Mapper class named "WeatherMapper" which implements the Mapper interface. It has private fields for "day" and "temperature" of type Text and IntWritable respectively. The "map" method takes an Object key, a Text value, and a Context context, throwing an IOException. It splits the value by comma, checks if there are more than two fields, sets the day to the first field, sets the temperature to the second field, and writes them to the context.

```
GNU nano 4.8          WeatherAnalysis.java          Modified
import java.io.IOException;
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 WeatherAnalysis {
    public static class WeatherMapper extends Mapper<Object, Text, Text, IntWr>
        private Text day = new Text();
        private IntWritable temperature = new IntWritable();

        public void map(Object key, Text value, Context context) throws IOException {
            String[] fields = value.toString().split(",");
            if (fields.length > 2) {
                day.set(fields[0]);
                temperature.set(Integer.parseInt(fields[1]));
                context.write(day, temperature);
            }
        }
    }
}

^G Get Help      ^O Write Out      ^W Where Is      ^K Cut Text      ^J Justify
^X Exit         ^R Read File     ^\ Replace       ^U Paste Text   ^T To Spell
```

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) -d . WeatherAnalysis.java
parinagarg@Ubuntu:~$
```

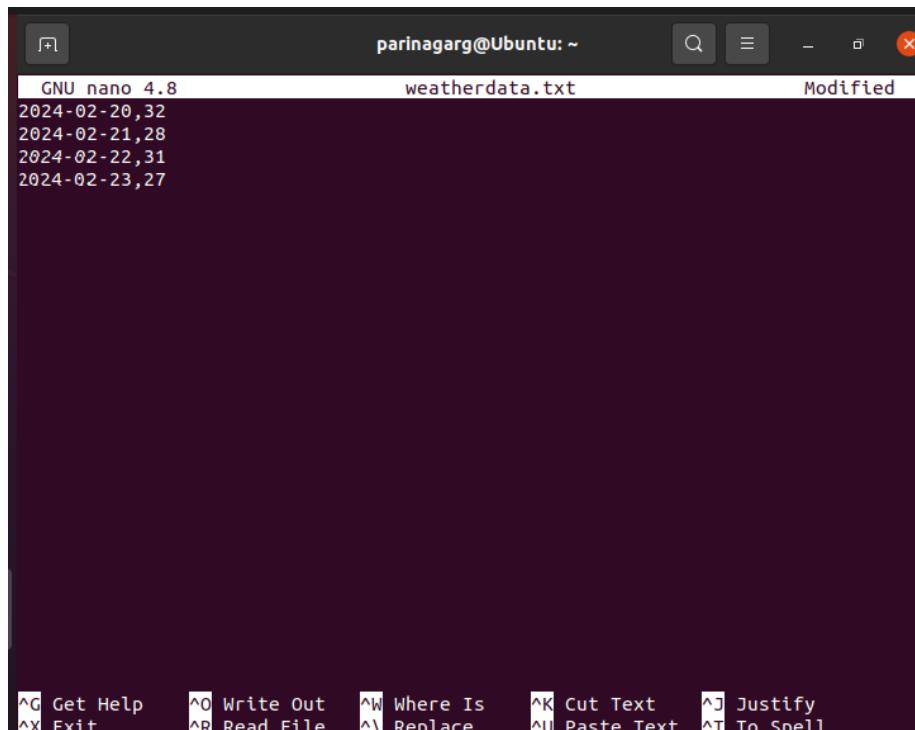
Step 4:

```
parinagarg@Ubuntu:~$ jar cvf wa.jar *.class
added manifest
adding: matrixMultiplication$MatrixMapper.class(in = 2297) (out= 990)(deflated 56%)
adding: MatrixMultiplication$MatrixMapper.class(in = 2297) (out= 989)(deflated 56%)
adding: matrixMultiplication$MatrixReducer.class(in = 2304) (out= 1053)(deflated 54%)
adding: MatrixMultiplication$MatrixReducer.class(in = 2304) (out= 1052)(deflated 54%)
adding: matrixMultiplication.class(in = 1609) (out= 843)(deflated 47%)
adding: MatrixMultiplication.class(in = 1609) (out= 842)(deflated 47%)
adding: MaxElectricConsumption$ConsumptionMapper.class(in = 1690) (out= 694)(deflated 58%)
adding: MaxElectricConsumption$ConsumptionReducer.class(in = 1781) (out= 752)(deflated 57%)
adding: MaxElectricConsumption.class(in = 1593) (out= 837)(deflated 47%)
adding: Max_temp$MaxTemperatureReducer.class(in = 1843) (out= 796)(deflated 56%)
adding: Max_temp$TemperatureMapper.class(in = 2533) (out= 1114)(deflated 56%)
adding: Max_temp.class(in = 1497) (out= 810)(deflated 45%)
adding: StudentGrades$GradeMapper.class(in = 2595) (out= 1185)(deflated 54%)
adding: StudentGrades$GradeReducer.class(in = 1526) (out= 635)(deflated 58%)
adding: StudentGrades.class(in = 1456) (out= 793)(deflated 45%)
adding: WeatherAnalysis$WeatherMapper.class(in = 1830) (out= 766)(deflated 58%)
adding: WeatherAnalysis$WeatherReducer.class(in = 1740) (out= 751)(deflated 56%)
adding: WeatherAnalysis.class(in = 1500) (out= 802)(deflated 46%)
adding: WordCount$IntSumReducer.class(in = 1755) (out= 750)(deflated 57%)
```

Step 5:

```
parinagarg@Ubuntu:~$ nano weatherdata.txt
parinagarg@Ubuntu:~$ █
```

Step 6:



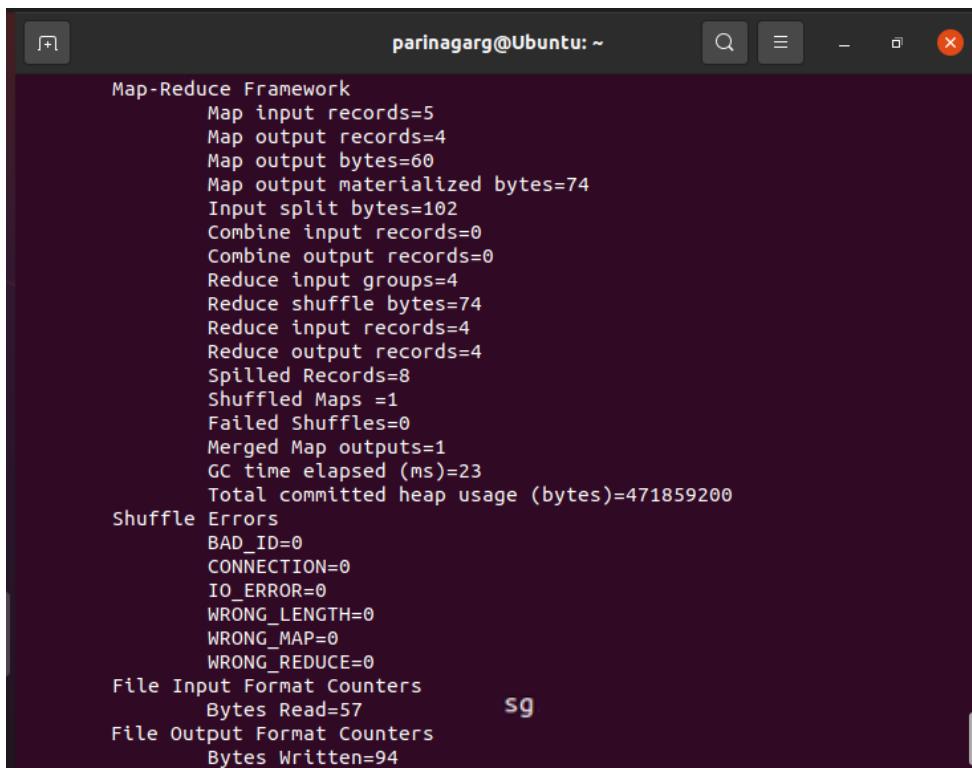
```
GNU nano 4.8
2024-02-20,32
2024-02-21,28
2024-02-22,31
2024-02-23,27
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar wa.jar WeatherAnalysis weatherdata.txt outputwa
2025-02-21 02:10:07,716 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2025-02-21 02:10:07,872 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot
period at 10 second(s).
2025-02-21 02:10:07,873 INFO impl.MetricsSystemImpl: JobTracker metrics system
started
2025-02-21 02:10:07,977 WARN mapreduce.JobResourceUploader: Hadoop command-line
option parsing not performed. Implement the Tool interface and execute your ap-
plication with ToolRunner to remedy this.
2025-02-21 02:10:08,116 INFO input.FileInputFormat: Total input files to proces-
s : 1
```

Step 8:



A screenshot of a terminal window titled "Map-Reduce Framework". The window displays various performance metrics for a job. Key values include:

- Map input records=5
- Map output records=4
- Map output bytes=60
- Map output materialized bytes=74
- Input split bytes=102
- Combine input records=0
- Combine output records=0
- Reduce input groups=4
- Reduce shuffle bytes=74
- Reduce input records=4
- Reduce output records=4
- Spilled Records=8
- Shuffled Maps =1
- Failed Shuffles=0
- Merged Map outputs=1
- GC time elapsed (ms)=23
- Total committed heap usage (bytes)=471859200

Under "Shuffle Errors", all categories (BAD_ID, CONNECTION, IO_ERROR, WRONG_LENGTH, WRONG_MAP, WRONG_REDUCE) are listed as 0.

Under "File Input Format Counters", Bytes Read=57.

Under "File Output Format Counters", Bytes Written=94.

Step 9:

```
parinagarg@Ubuntu:~$ hdfs dfs -cat outputwa/part-r-00000
2024-02-20      Shiny Day
2024-02-21      Cool Day
2024-02-22      Shiny Day
2024-02-23      Cool Day
parinagarg@Ubuntu:~$
```

Learning Outcomes:

EXPERIMENT-8

Title :- Develop a map reduce program to find the tags associated with each movie by analyzing movie lens data.

Introduction -

Code:

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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 MovieTagsAnalysis {

    public static class TagsMapper extends Mapper<Object, Text, Text, Text> {
        private Text movieID = new Text();
        private Text tag = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String[] fields = value.toString().split(",");
            if (fields.length >= 3) { // Ensuring proper data format
                movieID.set(fields[1]); // Movie ID is in the second column
                tag.set(fields[2]); // Tag is in the third column
                context.write(movieID, tag);
            }
        }
    }

    public static class TagsReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            List<String> tagsList = new ArrayList<>();
            for (Text val : values) {
                tagsList.add(val.toString());
            }
            context.write(key, new Text(String.join(", ", tagsList)));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Movie Tags Analysis");

        job.setJarByClass(MovieTagsAnalysis.class);
        job.setMapperClass(TagsMapper.class);
        job.setReducerClass(TagsReducer.class);
    }
}
```

```
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class);

FileInputFormat.addInputPath(job, new Path(args[0])); // Input directory
(tags.csv)
FileOutputFormat.setOutputPath(job, new Path(args[1])); // Output directory

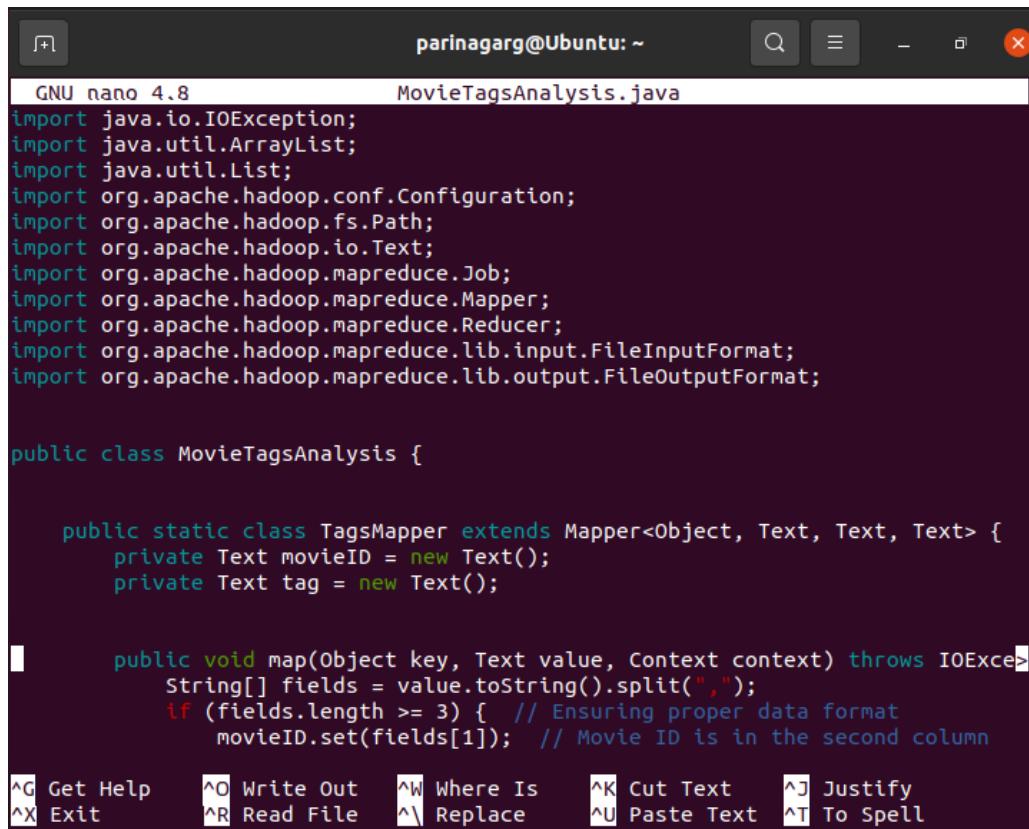
System.exit(job.waitForCompletion(true) ? 0 : 1);
}

}
```

Step 1:

```
parinagarg@Ubuntu:~$ nano MovieTagsAnalysis.java
parinagarg@Ubuntu:~$
```

Step 2:



```
GNU nano 4.8          MovieTagsAnalysis.java
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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 MovieTagsAnalysis {

    public static class TagsMapper extends Mapper<Object, Text, Text, Text> {
        private Text movieID = new Text();
        private Text tag = new Text();

        public void map(Object key, Text value, Context context) throws IOException {
            String[] fields = value.toString().split(",");
            if (fields.length >= 3) { // Ensuring proper data format
                movieID.set(fields[1]); // Movie ID is in the second column
                tag.set(fields[2]);
            }
        }

        protected void writeIntermediate(Text key, Text value, Context context) throws IOException {
            context.write(key, value);
        }
    }

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

        protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            int count = 0;
            for (Text value : values) {
                count++;
                result.set(value);
            }
            context.write(key, result);
        }
    }
}
```

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) MovieTagsAnalysis.java
parinagarg@Ubuntu:~$
```

Step 4:

```
parinagarg@Ubuntu:~$ jar cvf mta.jar MovieTagsAnalysis*.class
added manifest
adding: MovieTagsAnalysis$TagsMapper.class(in = 1644) (out= 694)(deflated 57%)
adding: MovieTagsAnalysis$TagsReducer.class(in = 1863) (out= 795)(deflated 57%)
adding: MovieTagsAnalysis.class(in = 1461) (out= 793)(deflated 45%)
parinagarg@Ubuntu:~$
```

Step 5:

```
parinagarg@Ubuntu:~$ nano tags.csv
```

Step 6:

The screenshot shows a terminal window titled "parinagarg@Ubuntu: ~". The command "nano tags.csv" has been run, displaying the following CSV data:

userId	movieId	tag	timestamp
1,100	Action	1256325400	
2,100	Thriller	1256325401	
3,101	Comedy	1256325402	
4,100	Adventure	1256325403	
5,102	Drama	1256325404	

The bottom status bar of the nano editor shows "[Wrote 6 lines]" and a series of keyboard shortcuts: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^X Exit, ^R Read File, ^\ Replace, ^U Paste Text, ^T To Spell.

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar mta.jar MovieTagsAnalysis tags.csv output_mta
2025-03-18 01:33:45,575 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2025-03-18 01:33:45,764 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot
period at 10 second(s).
2025-03-18 01:33:45,765 INFO impl.MetricsSystemImpl: JobTracker metrics system
started
2025-03-18 01:33:45,859 WARN mapreduce.JobResourceUploader: Hadoop command-line
option parsing not performed. Implement the Tool interface and execute your ap-
plication with ToolRunner to remedy this.
2025-03-18 01:33:45,942 INFO input.FileInputFormat: Total input files to proces
s : 1
```

Step 8:

```
Combine input records=0
Combine output records=0
Reduce input groups=4
Reduce shuffle bytes=96
Reduce input records=6
Reduce output records=4
Spilled Records=12
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=15
Total committed heap usage (bytes)=404750336
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=171
File Output Format Counters
    Bytes Written=84
```

Step 9:

```
parinagarg@Ubuntu:~$ hdfs dfs -cat output_mta/part-r-00000
movieId          tag
100      Adventure, Thriller, Action
101      Comedy
102      Drama
```

Learning Outcomes:

EXPERIMENT-9

Title :- Develop a map reduce program to analyze Uber data set to find the days on which each basement has more trips using the following data set. The uber data set consists of four columns they are:

Dispatching base number	Date	Active vehicle	Trips
--------------------------------	-------------	-----------------------	--------------

Introduction -

Code:

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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 UberTripsAnalysis {

    public static class UberMapper extends Mapper<Object, Text, Text, Text> {
        private Text baseId = new Text();
        private Text dateAndTrips = new Text();

        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            String[] tokens = value.toString().split("\\s+");
            // Split by spaces

            if (tokens.length == 4) { // Ensure the correct number of columns
                String base = tokens[0]; // Base ID
                String date = tokens[1]; // Date
                String trips = tokens[3]; // Number of trips

                baseId.set(base);
                dateAndTrips.set(date + "," + trips);
                context.write(baseId, dateAndTrips);
            }
        }
    }

    public static class UberReducer extends Reducer<Text, Text, Text, Text> {
        private Text maxTripDate = new Text();

        public void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
            int maxTripsValue = Integer.MIN_VALUE;
            String maxDay = "";

            for (Text val : values) {
                String[] parts = val.toString().split(",");
                String date = parts[0];
                int trips = Integer.parseInt(parts[1]);

                if (trips > maxTripsValue) {
                    maxTripsValue = trips;
                    maxDay = date;
                }
            }
        }
    }
}
```

```

        maxTripDate.set(maxDay + " " + maxTripsValue);
        context.write(key, maxTripDate);
    }
}

public static void main(String[] args) throws Exception {
    if (args.length < 2) {
        System.err.println("Usage: UberTripsAnalysis <input file> <output dir>");
        System.exit(1);
    }

    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Uber Trips Analysis");

    job.setJarByClass(UberTripsAnalysis.class);
    job.setMapperClass(UberMapper.class);
    job.setReducerClass(UberReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);

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

    FileInputFormat.addInputPath(job, new Path(args[0])); // Takes file directly
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

```

Step 1:

```
parinagarg@Ubuntu:~$ nano UberTripsAnalysis.java
parinagarg@Ubuntu:~$
```

Step 2:

The screenshot shows a terminal window titled "parinagarg@Ubuntu: ~". Inside the terminal, the file "UberTripsAnalysis.java" is being edited in the nano text editor. The code is a Java MapReduce program. It starts with imports for various org.apache.hadoop packages and java.io.IOException. The class UberTripsAnalysis contains a static inner class UberMapper that extends Mapper<Object, Text, Text, Text>. The map method splits the input value by whitespace and checks if it has four tokens. If so, it extracts the base ID and date from tokens[0] and tokens[1] respectively. The bottom of the terminal shows nano editor key bindings.

```
GNU nano 4.8           UberTripsAnalysis.java
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
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 UberTripsAnalysis {
    public static class UberMapper extends Mapper<Object, Text, Text, Text> {
        private Text baseId = new Text();
        private Text dateAndTrips = new Text();

        public void map(Object key, Text value, Context context) throws IOException {
            String[] tokens = value.toString().split("\\s+");
            if (tokens.length == 4) { // Ensure the correct number of columns
                String base = tokens[0]; // Base ID
                String date = tokens[1]; // Date
                [ Wrote 95 lines ]
            }
        }
    }
}
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) UberTripsAnalysis.java
parinagarg@Ubuntu:~$
```

Step 4:

```
parinagarg@Ubuntu:~$ jar cvf uta.jar UberTripsAnalysis*.class
added manifest
adding: UberTripsAnalysis$UberMapper.class(in = 2131) (out= 897)(deflated 57%)
adding: UberTripsAnalysis$UberReducer.class(in = 2406) (out= 1049)(deflated 56%)
adding: UberTripsAnalysis.class(in = 1721) (out= 928)(deflated 46%)
```

Step 5:

```
parinagarg@Ubuntu:~$ nano TripsData.txt
parinagarg@Ubuntu:~$
```

Step 6:

The screenshot shows a terminal window titled "parinagarg@Ubuntu:~". The file "TripsData.txt" is open in the nano editor. The content of the file is as follows:

```
GNU nano 4.8
B02512 2014-04-01 190 1000
B02512 2014-04-02 200 1283
B02512 2014-04-03 210 1200
B02598 2014-04-01 180 900
B02598 2014-04-02 190 1100
B02598 2014-04-03 195 1200
```

The status bar at the bottom of the terminal indicates "[Wrote 6 lines]". The keyboard shortcuts for nano editor are displayed at the bottom:

- ^G Get Help
- ^O Write Out
- ^W Where Is
- ^K Cut Text
- ^J Justify
- ^X Exit
- ^R Read File
- ^V Replace
- ^U Paste Text
- ^T To Spell

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar uta.jar UberTripsAnalysis TripsData.txt output_uta
2025-03-22 11:10:51,458 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2025-03-22 11:10:51,623 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2025-03-22 11:10:51,623 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2025-03-22 11:10:51,717 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.
2025-03-22 11:10:51,812 INFO input.FileInputFormat: Total input files to process : 1
```

Step 8:

```
parinagarg@Ubuntu: ~
Map input records=6
Map output records=6
Map output bytes=137
Map output materialized bytes=155
Input split bytes=100
Combine input records=0
Combine output records=0
Reduce input groups=2
Reduce shuffle bytes=155
Reduce input records=6
Reduce output records=2
Spilled Records=12
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=10
Total committed heap usage (bytes)=379584512
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=161
File Output Format Counters
Bytes Written=58
parinagarg@Ubuntu:~$
```

Step 9:

```
parinagarg@Ubuntu:~$ hdfs dfs -cat output_uta/part-r-00000
B02512 2014-04-02 1283
B02598 2014-04-03 1200
```

Learning Outcomes:

EXPERIMENT-10

Title :- Develop a map reduce program to analyze titanic dataset to find the average age of the people (both male and female) who died in the tragedy. How many people survived in each class.

Introduction -

Code:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;

public class TitanicAnalysis {

    // Mapper Class
    public static class TitanicMapper extends Mapper<LongWritable, Text, Text, Text> {
        public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
            String line = value.toString();
            String[] fields = line.split(",");
            if (fields.length < 4 || fields[0].equals("Survived")) return; // Skip header
            String survived = fields[0].trim();
            String pclass = fields[1].trim();
            String gender = fields[2].trim();
            String age = fields[3].trim();
            if (!age.isEmpty() && !age.equals("NA")) {
                if (survived.equals("0")) {
                    context.write(new Text("Deceased_" + gender), new Text(age));
                } else {
                    context.write(new Text("Survived_Class_" + pclass), new Text("1"));
                }
            }
        }
    }

    // Reducer Class
    public static class TitanicReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            int count = 0;
            double sum = 0.0;
            for (Text val : values) {
                try {
                    double num = Double.parseDouble(val.toString());
                    sum += num;
                    count++;
                } catch (NumberFormatException e) {
                    count++;
                }
            }
        }
    }
}
```

```

        if (key.toString().startsWith("Deceased")) {
            double avgAge = count == 0 ? 0 : sum / count;
            context.write(key, new Text("Average Age: " + avgAge));
        } else {
            context.write(key, new Text("Survivors: " + count));
        }
    }

// Driver Code
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "Titanic Analysis");
    job.setJarByClass(TitanicAnalysis.class);
    job.setMapperClass(TitanicMapper.class);
    job.setReducerClass(TitanicReducer.class);
    job.setOutputKeyClass(Text.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);
}
}

```

Step 1:

```
parinagarg@Ubuntu:~$ nano TitanicAgeAnalysis.java
parinagarg@Ubuntu:~$
```

Step 2:

The screenshot shows a terminal window with the title "parinagarg@Ubuntu: ~". The file "TitanicAgeAnalysis.java" is open in the nano text editor. The code implements a Mapper class named "TitanicMapper" that processes lines from a CSV file. It filters out rows where the age is missing ("NA") or zero ("0"). The terminal also displays various keyboard shortcuts for the nano editor.

```
GNU nano 4.8           TitanicAgeAnalysis.java           Modified
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import java.io.IOException;

public class TitanicAgeAnalysis {

    // Mapper Class
    public static class TitanicMapper extends Mapper<LongWritable, Text, Text, >
        public void map(LongWritable key, Text value, Context context) throws >
            String line = value.toString();
            String[] fields = line.split(",");
            if (fields.length < 4 || fields[0].equals("Survived")) return; //>

            String survived = fields[0].trim();
            String pclass = fields[1].trim();
            String gender = fields[2].trim();
            String age = fields[3].trim();

            if (!age.isEmpty() && !age.equals("NA")) {
                if (survived.equals("0")) {
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify
^X Exit ^R Read File ^\ Replace ^U Paste Text ^T To Spell

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) TitanicAgeAnalysis.java
parinagarg@Ubuntu:~$
```

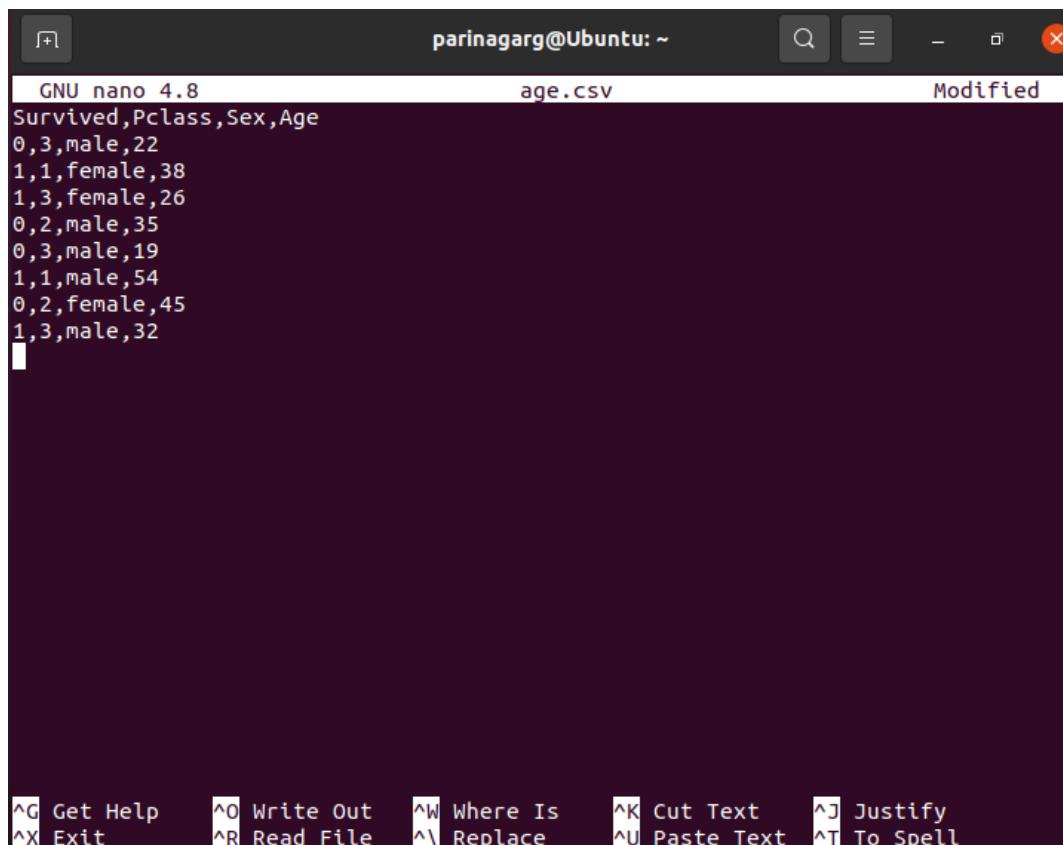
Step 4:

```
parinagarg@Ubuntu:~$ jar cvf taa.jar TitanicAgeAnalysis*.class
added manifest
adding: TitanicAgeAnalysis$TitanicMapper.class(in = 2398) (out= 1030)(deflated
57%)
adding: TitanicAgeAnalysis$TitanicReducer.class(in = 2484) (out= 1091)(deflated
56%)
adding: TitanicAgeAnalysis.class(in = 1474) (out= 793)(deflated 46%)
```

Step 5:

```
parinagarg@Ubuntu:~$ nano age.csv  
parinagarg@Ubuntu:~$
```

Step 6:



The screenshot shows the nano text editor interface. The title bar says "parinagarg@Ubuntu: ~". The status bar indicates "Modified". The main area displays the following CSV data:

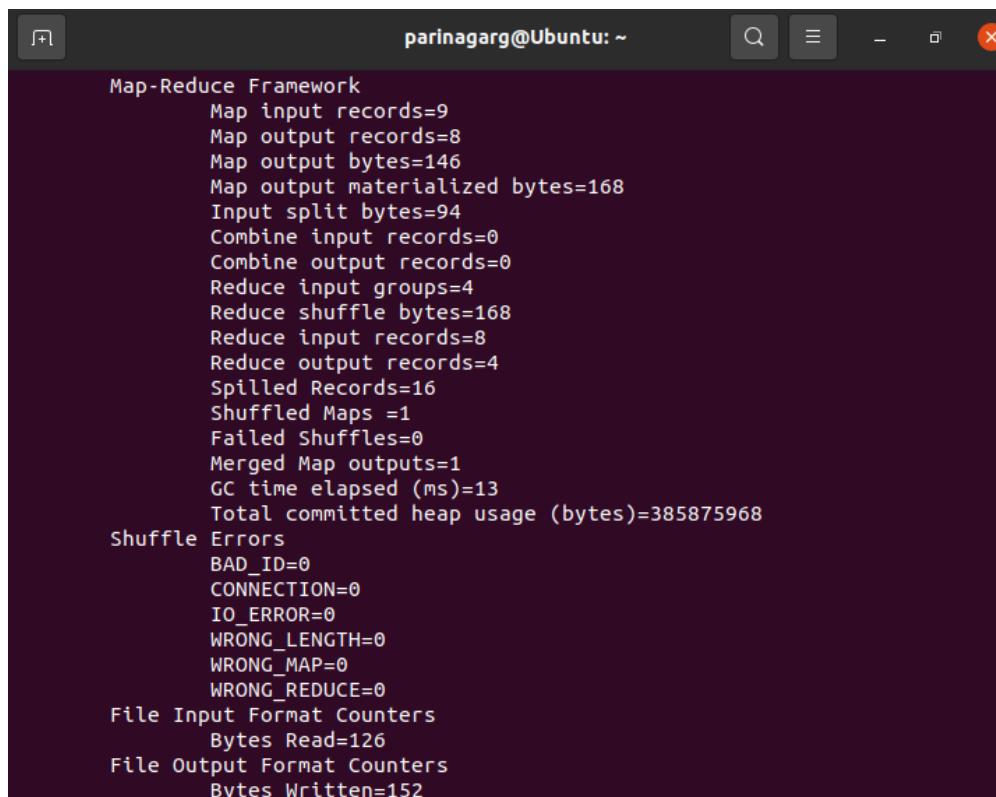
	Survived	Pclass	Sex	Age
0	3	male	22	
1	1	female	38	
1	3	female	26	
0	2	male	35	
0	3	male	19	
1	1	male	54	
0	2	female	45	
1	3	male	32	

At the bottom, there are several keyboard shortcut keys: ^G Get Help, ^O Write Out, ^W Where Is, ^K Cut Text, ^J Justify, ^X Exit, ^R Read File, ^\ Replace, ^U Paste Text, ^T To Spell.

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar taa.jar TitanicAgeAnalysis age.csv taa_output  
2025-03-18 10:35:02,158 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties  
2025-03-18 10:35:02,242 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).  
2025-03-18 10:35:02,243 INFO impl.MetricsSystemImpl: JobTracker metrics system started  
2025-03-18 10:35:02,302 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
2025-03-18 10:35:02,384 INFO input.FileInputFormat: Total input files to process : 1
```

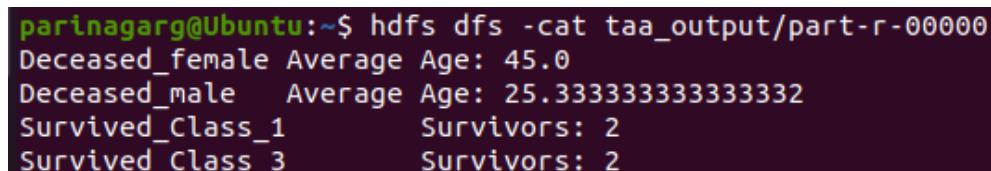
Step 8:



A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The window displays various MapReduce framework statistics. The output includes:

```
Map-Reduce Framework
  Map input records=9
  Map output records=8
  Map output bytes=146
  Map output materialized bytes=168
  Input split bytes=94
  Combine input records=0
  Combine output records=0
  Reduce input groups=4
  Reduce shuffle bytes=168
  Reduce input records=8
  Reduce output records=4
  Spilled Records=16
  Shuffled Maps =1
  Failed Shuffles=0
  Merged Map outputs=1
  GC time elapsed (ms)=13
  Total committed heap usage (bytes)=385875968
Shuffle Errors
  BAD_ID=0
  CONNECTION=0
  IO_ERROR=0
  WRONG_LENGTH=0
  WRONG_MAP=0
  WRONG_REDUCE=0
File Input Format Counters
  Bytes Read=126
File Output Format Counters
  Bytes Written=152
```

Step 9:



```
parinagarg@Ubuntu:~$ hdfs dfs -cat taa_output/part-r-00000
Deceased_female Average Age: 45.0
Deceased_male    Average Age: 25.333333333333332
Survived_Class_1   Survivors: 2
Survived Class 3   Survivors: 2
```

Learning Outcomes:

EXPERIMENT-11

Title :- Develop a map reduce program to find the maximum temperature in each month for each year.

Introduction -

Code:

```
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;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class MaxTemperatureMonth {
    public static class TempMapper extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context) throws IOException, InterruptedException {
            String[] fields = value.toString().split("\s+");
            if (fields.length == 3) {
                String year = fields[0];
                String month = fields[1];
                String temperature = fields[2];
                context.write(new Text(year), new Text(month + " " + temperature));
            }
        }
    }

    public static class TempReducer extends Reducer<Text, Text, Text, Text> {
        public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
            String maxMonth = "";
            int maxTemp = Integer.MIN_VALUE;

            for (Text val : values) {
                String[] parts = val.toString().split(" ");
                String month = parts[0];
                int temp = Integer.parseInt(parts[1]);

                if (temp > maxTemp) {
                    maxTemp = temp;
                    maxMonth = month;
                }
            }
            context.write(key, new Text(maxMonth + " " + maxTemp));
        }
    }

    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Max Temperature Month");
    }
}
```

```
job.setJarByClass(MaxTemperatureMonth.class);
    job.setMapperClass(TempMapper.class);
    job.setReducerClass(TempReducer.class);

    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

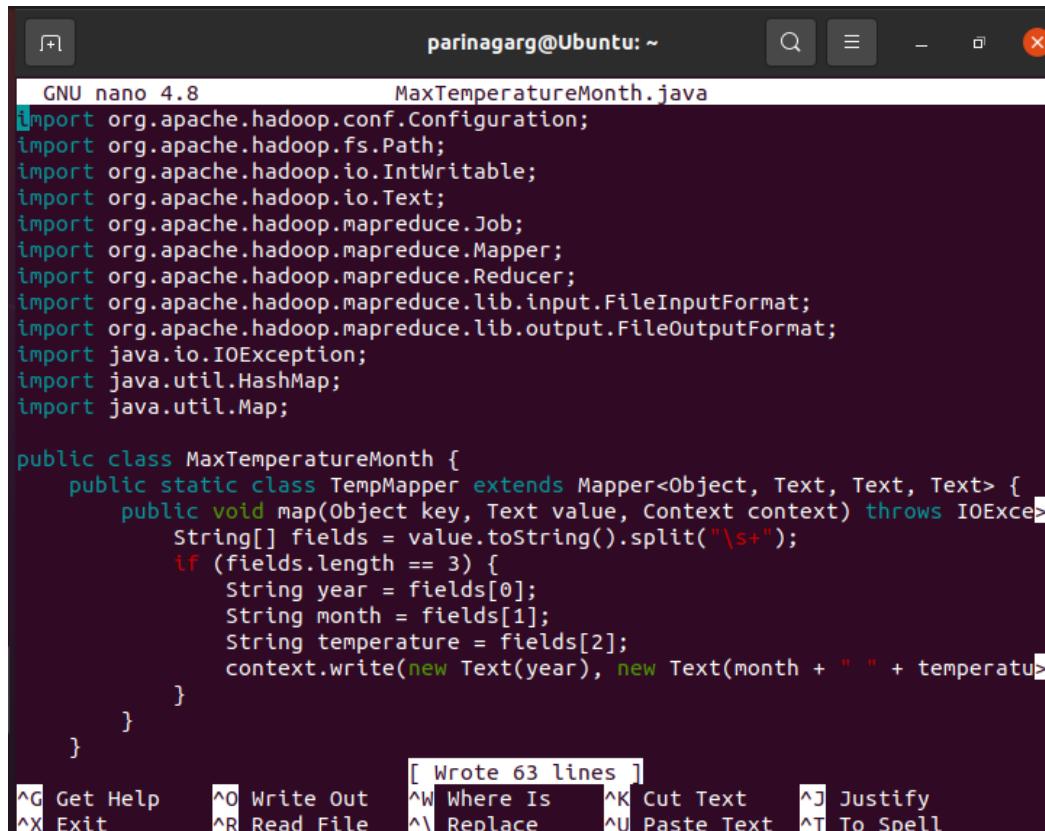
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

Step 1:

```
parinagarg@Ubuntu:~$ nano MaxTemperatureMonth.java
parinagarg@Ubuntu:~$
```

Step 2:



The screenshot shows a terminal window titled "parinagarg@Ubuntu: ~". It displays the Java code for "MaxTemperatureMonth.java". The code defines a class "MaxTemperatureMonth" with a static inner class "TempMapper" that implements the Mapper interface. The "map" method splits the input value by whitespace, extracts the year, month, and temperature, and writes them to context. The terminal shows the command "nano MaxTemperatureMonth.java" at the prompt, followed by the code itself. At the bottom, there are nano editor key bindings: ^G Get Help, ^O Write Out, [Wrote 63 lines], ^X Exit, ^R Read File, ^W Where Is, ^K Cut Text, ^J Justify, ^\ Replace, ^U Paste Text, ^T To Spell.

```
GNU nano 4.8          MaxTemperatureMonth.java
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;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

public class MaxTemperatureMonth {
    public static class TempMapper extends Mapper<Object, Text, Text, Text> {
        public void map(Object key, Text value, Context context) throws IOException {
            String[] fields = value.toString().split("\\s+");
            if (fields.length == 3) {
                String year = fields[0];
                String month = fields[1];
                String temperature = fields[2];
                context.write(new Text(year), new Text(month + " " + temperature));
            }
        }
    }
}
```

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) MaxTemperatureMonth.java
parinagarg@Ubuntu:~$
```

Step 4:

```
parinagarg@Ubuntu:~$ jar cvf mtm.jar MaxTemperatureMonth*.class
added manifest
adding: MaxTemperatureMonth$TempMapper.class(in = 1990) (out= 808)(deflated 59%)
adding: MaxTemperatureMonth$TempReducer.class(in = 2316) (out= 997)(deflated 56%)
adding: MaxTemperatureMonth.class(in = 1559) (out= 825)(deflated 47%)
```

Step 5:

```
parinagarg@Ubuntu:~$ nano maxtemp.txt  
parinagarg@Ubuntu:~$
```

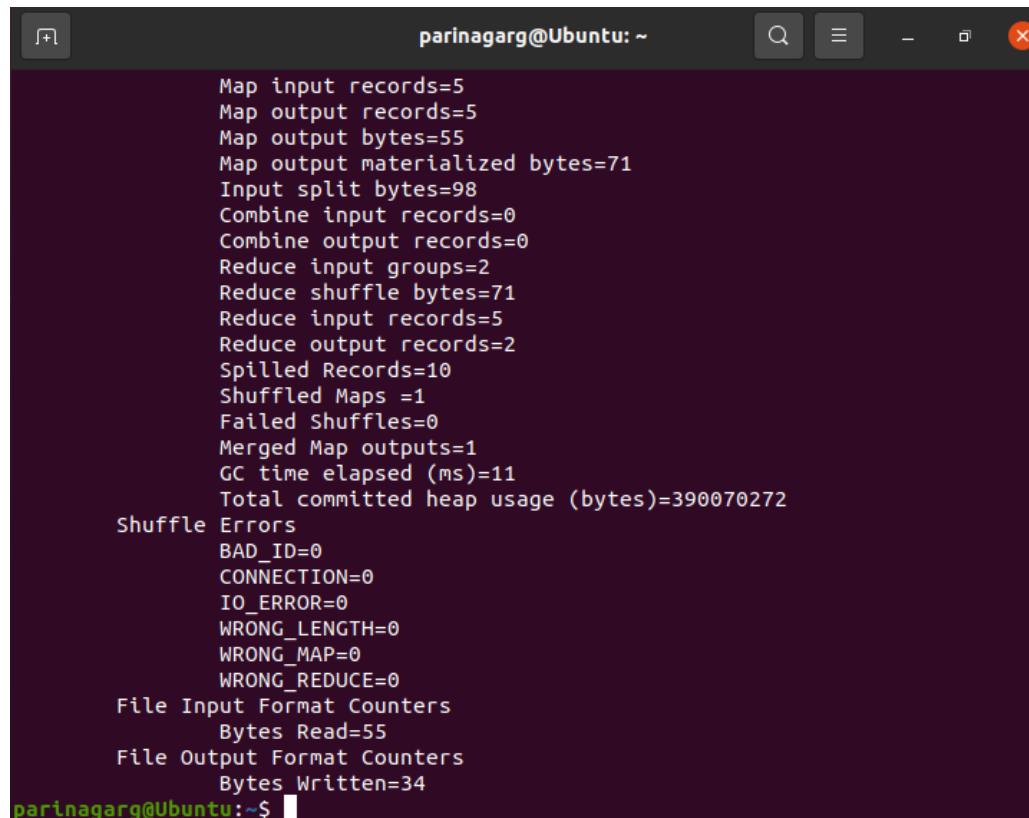
Step 6:

```
GNU nano 4.8  
2023 01 30  
2023 02 35  
2023 03 40  
2022 01 25  
2022 07 45  
[ Wrote 5 lines ]  
^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify  
^X Exit ^R Read File ^V Replace ^U Paste Text ^T To Spell
```

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar mtm.jar MaxTemperatureMonth maxtemp.txt output_mtm  
2025-03-25 10:09:58,759 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties  
2025-03-25 10:09:58,864 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).  
2025-03-25 10:09:58,864 INFO impl.MetricsSystemImpl: JobTracker metrics system started  
2025-03-25 10:09:58,941 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
2025-03-25 10:09:59,031 INFO input.FileInputFormat: Total input files to process : 1
```

Step 8:



A screenshot of a terminal window titled "parinagarg@Ubuntu: ~". The window displays various Hadoop job metrics. The output includes:

```
Map input records=5
Map output records=5
Map output bytes=55
Map output materialized bytes=71
Input split bytes=98
Combine input records=0
Combine output records=0
Reduce input groups=2
Reduce shuffle bytes=71
Reduce input records=5
Reduce output records=2
Spilled Records=10
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=11
Total committed heap usage (bytes)=390070272
shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=55
File Output Format Counters
Bytes Written=34
```

parinagarg@Ubuntu:~\$

Step 9:

```
parinagarg@Ubuntu:~$ hdfs dfs -cat output_mtmt/part-r-00000
2022    07 45
2023    03 40
```

Learning Outcomes:

EXPERIMENT-12

Title :- Develop a map reduce program to Sort and aggregate the data in Hadoop.

Introduction -

Code:

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

public class SortAggregate {

    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 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);
        }
    }

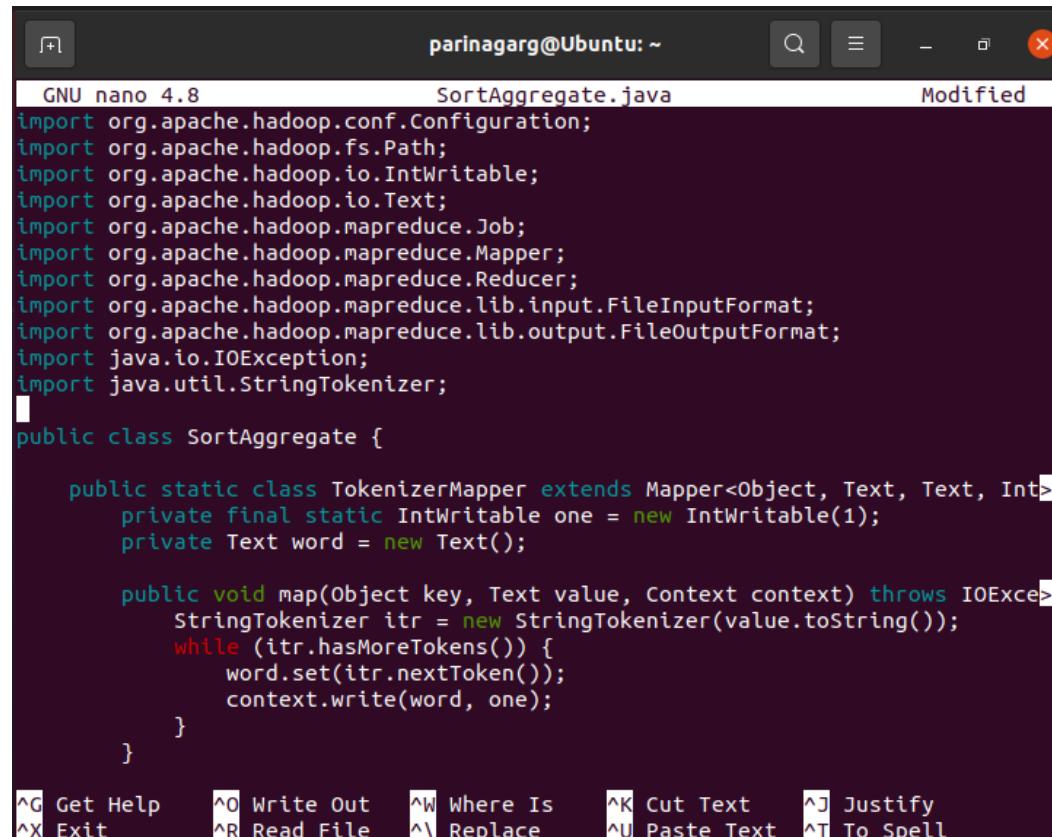
    public static void main(String[] args) throws Exception {
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "sort and aggregate");
        job.setJarByClass(SortAggregate.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        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);
}
}
```

Step 1:

```
parinagarg@Ubuntu:~$ nano SortAggregate.java
parinagarg@Ubuntu:~$
```

Step 2:



The screenshot shows the nano text editor window on a Linux desktop. The title bar says "parinagarg@Ubuntu: ~". The status bar at the bottom shows "Modified". The main area contains the Java code for SortAggregate. The code defines a Mapper class with a map method that tokenizes input and writes tokens to output. At the bottom of the editor, there is a menu of keyboard shortcuts.

```
GNU nano 4.8          SortAggregate.java          Modified
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;
import java.io.IOException;
import java.util.StringTokenizer;

public class SortAggregate {

    public static class TokenizerMapper extends Mapper<Object, Text, Text, Int>
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

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

    ^G Get Help      ^O Write Out     ^W Where Is      ^K Cut Text      ^J Justify
    ^X Exit         ^R Read File     ^L Replace       ^U Paste Text    ^T To Spell
```

Step 3:

```
parinagarg@Ubuntu:~$ javac -cp $(hadoop classpath) SortAggregate.java
parinagarg@Ubuntu:~$
```

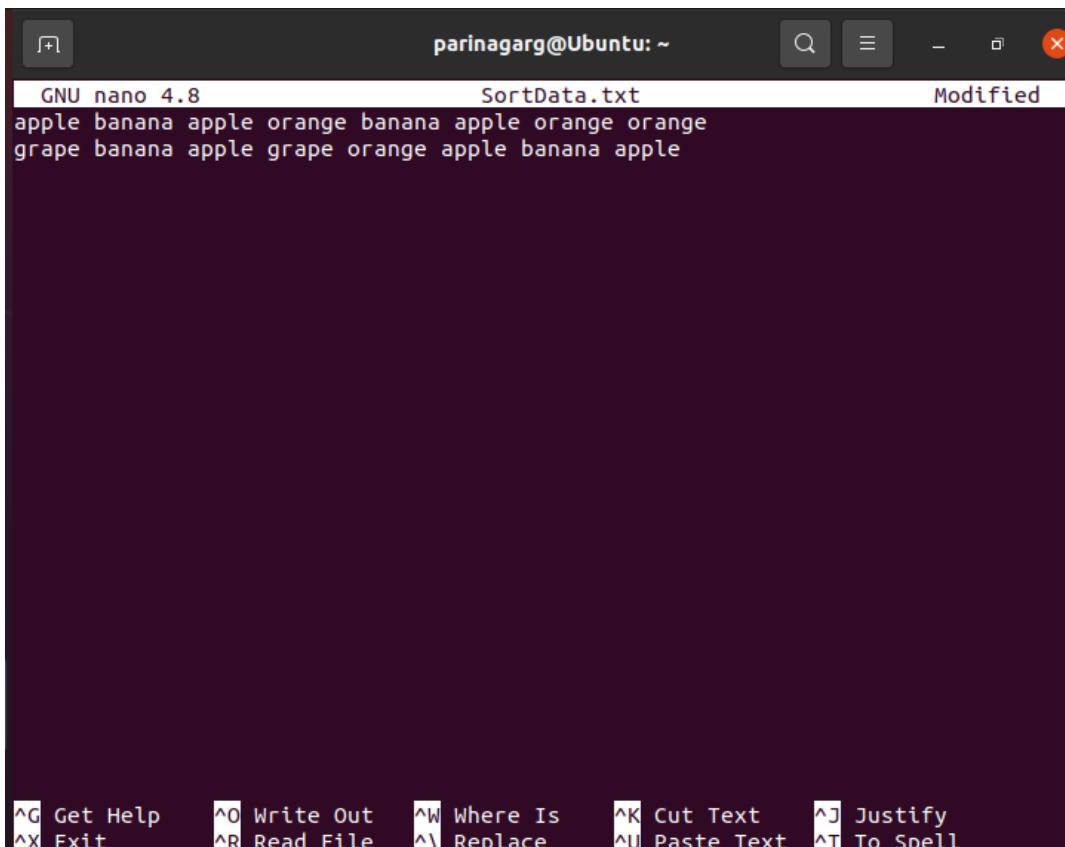
Step 4:

```
parinagarg@Ubuntu:~$ jar cvf sa.jar SortAggregate*.class
added manifest
adding: SortAggregate$IntSumReducer.class(in = 1767) (out= 756)(deflated 57%)
adding: SortAggregate$TokenizerMapper.class(in = 1764) (out= 766)(deflated 56%)
adding: SortAggregate.class(in = 1535) (out= 834)(deflated 45%)
parinagarg@Ubuntu:~$
```

Step 5:

```
parinagarg@Ubuntu:~$ nano SortData.txt
parinagarg@Ubuntu:~$
```

Step 6:



The screenshot shows the nano text editor window. The title bar says "parinagarg@Ubuntu: ~". The status bar indicates "GNU nano 4.8", "SortData.txt", and "Modified". The main area contains the following text:

```
apple banana apple orange banana apple orange orange
grape banana apple grape orange apple banana apple
```

At the bottom, there is a menu bar with icons for file operations like Open, Save, and Exit. Below the menu is a toolbar with icons for Help, Write Out, Where Is, Cut Text, Justify, Exit, Read File, Replace, Paste Text, and To Spell. The keyboard shortcuts for these commands are listed below the toolbar.

Keyboard shortcuts at the bottom:

- ^G Get Help
- ^O Write Out
- ^W Where Is
- ^K Cut Text
- ^J Justify
- ^X Exit
- ^R Read File
- ^A Replace
- ^U Paste Text
- ^T To Spell

Step 7:

```
parinagarg@Ubuntu:~$ hadoop jar sa.jar SortAggregate SortData.txt output_sa
2025-04-06 16:28:34,619 INFO impl.MetricsConfig: Loaded properties from hadoop-
metrics2.properties
2025-04-06 16:28:34,808 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot
period at 10 second(s).
2025-04-06 16:28:34,809 INFO impl.MetricsSystemImpl: JobTracker metrics system
started
2025-04-06 16:28:34,907 WARN mapreduce.JobResourceUploader: Hadoop command-line
option parsing not performed. Implement the Tool interface and execute your ap-
plication with ToolRunner to remedy this.
2025-04-06 16:28:35,006 INFO input.FileInputFormat: Total input files to proces-
s : 1
```

Step 8:

```
parinagarg@Ubuntu: ~
Map input records=2
Map output records=16
Map output bytes=168
Map output materialized bytes=56
Input split bytes=99
Combine input records=16
Combine output records=4
Reduce input groups=4
Reduce shuffle bytes=56
Reduce input records=4
Reduce output records=4
Spilled Records=8
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=18
Total committed heap usage (bytes)=383778816
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
    Bytes Read=104
File Output Format Counters
    Bytes Written=46
parinagarg@Ubuntu:~$
```

Step 9:

```
parinagarg@Ubuntu:~$ hdfs dfs -cat output_sa/part-r-00000
apple    6
banana   4
grape    2
orange   4
```

Learning Outcomes:
