1. Adding files and directories,Retrieving files,Deleting files and directories

2. Develop a MapReduce program to implement Matrix Multiplication

matrix ops

A,0,0,1

A,0,1,2

A,1,0,3

A,1,1,4

B,0,0,5

B,0,1,6

B,1,0,7

B,1,1,8

==========

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

public class MatrixDriver {
   public static void main(String[] args) throws Exception {
      if (args.length != 2) {
         System.err.println("Usage: MatrixDriver <input> <output>");
         System.exit(-1);
      }

      Configuration conf = new Configuration();
      Job job = new Job(conf, "Matrix Multiplication");

      job.setJarByClass(MatrixDriver.class);
      job.setMapperClass(MatrixMapper.class);
      job.setReducerClass(MatrixReducer.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);
    }
}
```

```
==========================================================================
=
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;

public class MatrixMapper extends Mapper<Object, Text, Text, Text> {
    public void map(Object key, Text value, Context context)
            throws IOException, InterruptedException {

        // Input format: MatrixName,i,j,value
        // Example: A,0,0,2
        String[] parts = value.toString().split(",");
        String matrix = parts[0];
        int i = Integer.parseInt(parts[1]);
        int j = Integer.parseInt(parts[2]);
        int val = Integer.parseInt(parts[3]);

        int N = 2;  // assume 2x2 matrices for simplicity

        if (matrix.equals("A")) {
            // A[i][k] goes to all C[i][j]
            for (int col = 0; col < N; col++) {
                context.write(new Text(i + "," + col),
                        new Text("A," + j + "," + val));
            }
        } else {
            // B[k][j] goes to all C[i][j]
            for (int row = 0; row < N; row++) {
                context.write(new Text(row + "," + j),
                        new Text("B," + i + "," + val));
            }
        }
    }
}
```

```
================================================
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import java.io.IOException;

public class MatrixReducer extends Reducer<Text, Text, Text, Text> {
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
```

```
        // For simplicity, assume max 10 values
        int[] Aval = new int[10];
        int[] Bval = new int[10];

        for (Text t : values) {
            String[] parts = t.toString().split(",");
            String m = parts[0];
            int index = Integer.parseInt(parts[1]);
            int val = Integer.parseInt(parts[2]);

            if (m.equals("A")) {
                Aval[index] = val;
            } else {
                Bval[index] = val;
            }
        }

        int sum = 0;
        for (int k = 0; k < 10; k++) {
            sum += Aval[k] * Bval[k];
        }

        context.write(key, new Text(Integer.toString(sum)));
    }
}
```

3. Develop a Map Reduce program that mines weather data and displays appropriate messages indicating the weather conditions of the day.

Date,TempC,Humidity,WindKmph,PrecipMM
2025-08-20,36,55,12,0
2025-08-20,34,88,15,1
2025-08-21,28,60,45,0
2025-08-21,24,82,18,3
2025-08-22,6,70,8,0
2025-08-22,2,65,10,0
2025-08-23,30,50,10,22
2025-08-24,31,85,8,0

Classification rules (edit freely)

1. PrecipMM >= 20 → **Heavy Rain** (severity 6)
2. PrecipMM >= 2 → **Rainy** (severity 5)
3. TempC >= 35 → **Hot** (severity 4)

4. TempC <= 5 → **Cold** (severity 3)
5. WindKmph >= 40 → **Windy** (severity 2)
6. Humidity >= 80 → **Humid** (severity 1)
7. else → **Clear** (severity 0)

==========

```
import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;


public class WeatherConditionsMapper extends Mapper<LongWritable, Text, Text, Text> {

@Override

protected void map(LongWritable key, Text value, Context context) throws java.io.IOExcepti
on, InterruptedException {

String line = value.toString().trim();

if (line.isEmpty()) return;

// Skip header if present

if (line.toLowerCase().startsWith("date,")) return;


String[] parts = line.split(",");

if (parts.length < 5) return; // bad row


String date = parts[0].trim();

try {

double tempC = Double.parseDouble(parts[1].trim());

double humidity = Double.parseDouble(parts[2].trim());

double wind = Double.parseDouble(parts[3].trim());

double precip = Double.parseDouble(parts[4].trim());


// Compute severity + message

int severity; String message;

if (precip >= 20) { severity = 6; message = "Heavy Rain"; }
```

```
else if (precip >= 2) { severity = 5; message = "Rainy"; }
else if (tempC >= 35) { severity = 4; message = "Hot"; }
else if (tempC <= 5) { severity = 3; message = "Cold"; }
else if (wind >= 40) { severity = 2; message = "Windy"; }
else if (humidity >= 80) { severity = 1; message = "Humid"; }
else { severity = 0; message = "Clear"; }

// Emit: key = date, value = "severity|message"
context.write(new Text(date), new Text(severity + "|" + message));
} catch (NumberFormatException e) {
// skip invalid numeric rows
}
}
}
```

==========

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class WeatherConditionsReducer extends Reducer<Text, Text, Text, Text> {
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) throws java.io.IOEx
ception, InterruptedException {
int bestSeverity = -1;
String bestMessage = "";

for (Text t : values) {
String[] kv = t.toString().split("\\|");
if (kv.length != 2) continue;
try {
int sev = Integer.parseInt(kv[0]);
String msg = kv[1];
```

```
if (sev > bestSeverity) { bestSeverity = sev; bestMessage = msg; }
} catch (NumberFormatException ignored) {}
}

// Output: Date \t Message
if (bestSeverity >= 0) {
context.write(key, new Text(bestMessage));
}
}
}
```

==========

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

public class WeatherConditionsDriver {
public static void main(String[] args) throws Exception {
if (args.length != 2) {
System.err.println("Usage: WeatherConditionsDriver <input> <output>");
System.exit(-1);
}

Configuration conf = new Configuration();
Job job = new Job(conf, "Daily Weather Conditions");
job.setJarByClass(WeatherConditionsDriver.class);

job.setMapperClass(WeatherConditionsMapper.class);
```

```
job.setReducerClass(WeatherConditionsReducer.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);
}
}
```

4. Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.

movie lens

```
 userId,movieId,tag,timestamp
15,1193,good plot,16234567
15,1193,classic,16234570
20,1200,funny,16234600
35,1200,boring,16234620
35,1193,emotional,16234625
```

==========

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
```

```
import org.apache.hadoop.mapreduce.Mapper;


public class TagsMapper extends Mapper<LongWritable, Text, Text, Text> {
   @Override
   protected void map(LongWritable key, Text value, Context context)
         throws java.io.IOException, InterruptedException {
      String line = value.toString().trim();
      if (line.isEmpty()) return;


      // Skip header
      if (line.toLowerCase().startsWith("userid")) return;


      String[] parts = line.split(",", 4);
      if (parts.length < 3) return;


      String movieId = parts[1].trim();
      String tag = parts[2].trim();


      if (!movieId.isEmpty() && !tag.isEmpty()) {
         context.write(new Text(movieId), new Text(tag));
      }
   }
}
```

==========

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

```
import org.apache.hadoop.mapreduce.Reducer;
import java.util.Iterator;

public class TagsReducer extends Reducer<Text, Text, Text, Text> {
   @Override
   protected void reduce(Text key, Iterable<Text> values, Context context)
         throws java.io.IOException, InterruptedException {
      StringBuilder sb = new StringBuilder();
      Iterator<Text> it = values.iterator();

      while (it.hasNext()) {
         sb.append(it.next().toString());
         if (it.hasNext()) sb.append(", ");
      }

      context.write(key, new Text(sb.toString()));
   }
}
```

==========

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

public class TagsDriver {
   public static void main(String[] args) throws Exception {
      if (args.length != 2) {
```

```
            System.err.println("Usage: TagsDriver <input> <output>");
            System.exit(-1);
        }

        Configuration conf = new Configuration();
        Job job = new Job(conf, "Movie Tags Extraction");
        job.setJarByClass(TagsDriver.class);

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

5. Implement Functions: Count – Sort – Limit – Skip – Aggregate using MongoDB

mongo db lab

```
db.data.insertMany([
  { _id: 1, name: 'John Doe', age: 30, department: 'HR', salary: 60000 },
    { _id: 2, name: 'Jane Smith', age: 25, department: 'Engineering', salary: 80000 },
    { _id: 3, name: 'Sam Johnson', age: 45, department: 'Engineering', salary: 120000 },
    { _id: 4, name: 'Chris Lee', age: 35, department: 'Marketing', salary: 75000 },
```

```
    { _id: 5, name: 'Emma Brown', age: 29, department: 'HR', salary: 65000 },
    { _id: 6, name: 'Alex Taylor', age: 32, department: 'Engineering', salary: 95000 },
    { _id: 7, name: 'Sophia Williams', age: 28, department: 'Marketing', salary: 70000 },
    { _id: 8, name: 'James Davis', age: 50, department: 'Management', salary: 150000 }
])

db.data.find().sort({ age: 1 })
db.data.find().limit(3)
db.data.count({ department: "Engineering" })

db.data.aggregate([
  { $group: { _id: "$department", totalEmployees: { $sum: 1 } } }
])



db.data.aggregate([
  { $group: { _id: "$department", averageSalary: { $avg: "$salary" } } }
])
```

6. Write Pig Latin scripts to sort, group, join, project, and filter the data.

first create people_data.txt file

```
Alice,30,New York
Bob,25,California
```

Charlie,35,Texas

David,30,California

Eva,20,New York

Frank,40,California

Grace,30,Texas

Hannah,45,New York

Ivy,25,California

Jack,20,Texas

---

copy data to hadoop directory

> hadoop fs -put people_data.txt people_data.txt

---

write pig script

pig1.pig

people= LOAD 'people_data.txt' USING PigStorage(',')as (name:chararray, age:int, city:chararray);

dump people;

---

pig2.pig

people= LOAD 'people_data.txt' USING PigStorage(',')as (name:chararray, age:int, city:chararray);

filtered_data = FILTER people BY age > 30;

dump filtered_data;

---

pig3.pig

people= LOAD 'people_data.txt' USING PigStorage(',')as (name:chararray, age:int, city:chararray);

Group_data = GROUP people by city;

dump Group_data;

---

pig4.pig

people= LOAD 'people_data.txt' USING PigStorage(',')as (name:chararray, age:int, city:chararray);

sort_age = ORDER people BY age DESC;

dump sort_age;

---

pig5.pig

people_data.txt:

sql
Copy code
John, 30, New York
Alice, 25, Los Angeles
Bob, 35, Chicago
Charlie, 28, New York

city_data.txt:

sql
Copy code
New York, 8000000, New York
Los Angeles, 4000000, California
Chicago, 2700000, Illinois
San Francisco, 870000, California

Now, let's perform the join on the city field.

Pig Script:

```
-- Load the people data
people = LOAD 'people_data.txt' USING PigStorage(',') AS (name:chararray, age:int,
city:chararray);

-- Load the city data
city_data = LOAD 'city_data.txt' USING PigStorage(',') AS (city:chararray, population:int,
state:chararray);

-- Perform the join on the 'city' field
joined_data = JOIN people BY city, city_data BY city;

-- Display the result of the join
DUMP joined_data;
```

7.Use Hive to create, alter, and drop databases, tables, views, functions, and indexes.

HIVE LAB

---

1. create data base

hive > CREATE DATABASE db1;

2. show data bases

hive> show databases;

3. drop data base

```
hive> drop database db1;
```

4. show data bases

```
hive> show databases;
```

5.create database and alter database owner

```
hive > create database db1;
```

```
hive>ALTER DATABASE db1 SET DBPROPERTIES ('owner'='John');
```

6. create emp table in db1 as below

```
HIVE> create database db1;
hive > use db1;
```

```
hive> CREATE TABLE employees (
  emp_id INT,
  emp_name STRING,
  emp_age INT,
  emp_city STRING,
  emp_dept STRING,
  emp_sal  INT
)
row format delimited
fields terminated by ','
```

lines terminated by '\n'

stored as textfile;

$cat > employees.txt

101,scott,34,chennai,hr,45000

102,lara,32,bangalore,fin,55000

103,john,41,hyd,mark,56000

104,michel,35,chennai,hr,67000

105,clark,28,bangalore,fin,86000

ctrl+z

hive>load data local inpath 'employees.txt' overwrite into employees;

$ cat > hivescript1.ql

$hive -S -f hivescript1.ql

7. emp salary greater than 50000

```
hive> select * from employees where emp_sal > 50000;
```

8. create view for your hive table

```
CREATE VIEW AGE_gt_30 AS
SELECT emp_id, emp_name, emp_age, emp_city
FROM employees
WHERE emp_age > 30;
```

9.execute view

```
SELECT * FROM AGE_gt_30;
```

10. drop the view

```
DROP VIEW IF EXISTS AGE_gt_30;
```

11. create index

```
CREATE INDEX emp_city_index
ON TABLE employees (emp_city)
AS 'COMPACT'
WITH DEFERRED REBUILD;
```

12. drop index

DROP INDEX IF EXISTS emp_city_index ON employees;

Implement a word count program in Hadoop and Spark.

Google colab hadoop pyspark

```
# Step 1: Install PySpark and Hadoop dependencies

!apt-get install openjdk-8-jdk-headless -qq > /dev/null

!wget -q http://apache.mirrors.lucidnetworks.net/spark/spark-3.1.2/spark-3.1.2-bin-hadoop3.2.tgz

!tar xvf spark-3.1.2-bin-hadoop3.2.tgz

!pip install findspark


# Step 2: Set environment variables

import os

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"

os.environ["SPARK_HOME"] = "/content/spark-3.1.2-bin-hadoop3.2"


# Step 3: Initialize PySpark

import findspark

findspark.init()


from pyspark.sql import SparkSession
```

```
# Create the Spark session
spark = SparkSession.builder.master("local").appName("WordCount").getOrCreate()

# Step 4: Create a sample text input (simulating a file)
input_data = [
    "Hello Spark",
    "This is a test for Word Count",
    "Spark is great for big data processing",
    "Word Count is a simple example"
]

# Parallelize the list to create an RDD
rdd = spark.sparkContext.parallelize(input_data)

# Step 5: Perform Word Count (MapReduce)
word_pairs = rdd.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1))
word_count = word_pairs.reduceByKey(lambda x, y: x + y)

# Step 6: Collect and print results
result = word_count.collect()
for word, count in result:
    print(f"{word}: {count}")

# Step 7: Save the results to a local file (simulating HDFS save)
word_count.saveAsTextFile("/content/word_count_output")
```