Week-1

Implement the following Data structures in Java a) Linked Lists b) Stacks c) Queues d) Set e) Map

Programs:

a)Linked list

```
import java.util.*;
public class Linkedlst{
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      LinkedList<String> list = new LinkedList<>();
      System.out.println("Linked List Operations:");
      System.out.println("1. Add Elements\n2. Remove Element\n3. Display Elements\n4. Exit");
      while (true) {
         System.out.print("Enter your choice: ");
         int choice = scanner.nextInt();
         scanner.nextLine();
         switch (choice) {
            case 1:
               System.out.print("Enter elements to add (comma-separated): ");
               String[] elements = scanner.nextLine().split(",");
               for (String element : elements) {
                  list.add(element.trim());
               }
               break;
            case 2:
               System.out.print("Enter element to remove: ");
               String element = scanner.nextLine();
               list.remove(element);
```

```
                break;
            case 3:
                System.out.println("Elements: " + list);
                break;
            case 4:
                return;
            default:
                System.out.println("Invalid choice.");}
        }
    }
}
```

output:

```
C:\Users\HEMANTH KUMAR\OneDrive\Desktop\New folder>java Linkedlst
Linked List Operations:
1. Add Elements
2. Remove Element
3. Display Elements
4. Exit
Enter your choice: 1
Enter elements to add (comma-separated): 1,2,3
Enter your choice: 3
Elements: [1, 2, 3]
Enter your choice: 2
Enter element to remove: 3
Enter your choice: 3
Elements: [1, 2]
Enter your choice: 4
```

b) Stack:

```
import java.util.*;
public class Stackprg{
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Stack<String> stack = new Stack<>();
        System.out.println("Stack Operations:");
```

```
        System.out.println("1. Push Elements\n2. Pop Element\n3. Peek Element\n4. Display Elements\n5. Exit");
        while (true) {
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();
            scanner.nextLine();
            switch (choice) {
                case 1:
                    System.out.print("Enter elements to push (comma-separated): ");
                    String[] elements = scanner.nextLine().split(",");
                    for (String element : elements) {
                        stack.push(element.trim());
                    }
                    break;
                case 2:
                    if (!stack.isEmpty()) {
                        System.out.println("Popped: " + stack.pop());
                    } else {
                        System.out.println("Stack is empty.");
                    }
                    break;
                case 3:
                    if (!stack.isEmpty()) {
                        System.out.println("Top element: " + stack.peek());
                    } else {
                        System.out.println("Stack is empty.");
                    }
                    break;
                case 4:
                    System.out.println("Elements: " + stack);
                    break;
```

```
            case 5:
                return;
            default:
                System.out.println("Invalid choice.");
        }
    }
  }
}
```

Actual output:

```
C:\Users\HEMANTH KUMAR\OneDrive\Desktop\New folder>java StackProgram
Stack Operations:
1. Push Elements
2. Pop Element
3. Peek Element
4. Display Elements
5. Display Stack Size
6. Clear Stack
7. Exit
Enter your choice: 1
Enter elements to push (comma-separated): 1,2,3,4
Elements pushed successfully.
Enter your choice: 2
Popped: 4
Enter your choice: 4
Elements: [1, 2, 3]
Enter your choice: 7
Exiting the program.
```

c) Queue

```
import java.util.*;
public class Queue{
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      Queue<String> queue = new LinkedList<>();
      System.out.println("Queue Operations:");
      System.out.println("1. Enqueue Elements\n2. Dequeue Element\n3. Display
Elements\n4. Exit");
```

```
    while (true) {
        System.out.print("Enter your choice: ");
        int choice = scanner.nextInt();
        scanner.nextLine();
        switch (choice) {
            case 1:
                System.out.print("Enter elements to enqueue (comma-separated): ");
                String[] elements = scanner.nextLine().split(",");
                for (String element : elements) {
                    queue.add(element.trim());
                }
                break;
            case 2:
                if (!queue.isEmpty()) {
                    System.out.println("Dequeued: " + queue.poll());
                } else {
                    System.out.println("Queue is empty.");
                }
                break;
            case 3:
                System.out.println("Elements: " + queue);
                break;
            case 4:
                return;
            default:
                System.out.println("Invalid choice.");
        }
    }
}
```

Actual output:

```
C:\Users\HEMANTH KUMAR\OneDrive\Desktop\New folder>java Queueprg
Queue Operations:
1. Enqueue Elements
2. Dequeue Element
3. Display Elements
4. Exit
Enter your choice: 1
Enter elements to enqueue (comma-separated): 1,2,3,4
Enter your choice: 3
Elements: [1, 2, 3, 4]
Enter your choice: 2
Dequeued: 1
Enter your choice: 4
```

d) Set

```
import java.util.*;
public class sets{
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
      Set<String> set = new HashSet<>();
      System.out.println("Set Operations:");
      System.out.println("1. Add Elements\n2. Remove Element\n3. Display Elements\n4. Exit");
      while (true) {
         System.out.print("Enter your choice: ");
         int choice = scanner.nextInt();
         scanner.nextLine();
         switch (choice) {
            case 1:
               System.out.print("Enter elements to add (comma-separated): ");
               String[] elements = scanner.nextLine().split(",");
```

```
            for (String element : elements) {
                set.add(element.trim());
            }
            break;
        case 2:
            System.out.print("Enter element to remove: ");
            String element = scanner.nextLine();
            set.remove(element);
            break;
        case 3:
            System.out.println("Elements: " + set);
            break;
        case 4:
            return;
        default:
            System.out.println("Invalid choice.");
      }
    }
  }
}
```



```
C:\Users\HEMANTH KUMAR\OneDrive\Desktop\New folder>java sets
Set Operations:
1. Add Elements
2. Remove Element
3. Display Elements
4. Exit
Enter your choice: 1
Enter elements to add (comma-separated): 1,2,3,4
Enter your choice: 3
Elements: [1, 2, 3, 4]
Enter your choice: 2
Enter element to remove: 2
Enter your choice: 3
Elements: [1, 3, 4]
Enter your choice: 4
```

e) Map

```
import java.util.*;

public class map {

   public static void main(String[] args) {

      Scanner scanner = new Scanner(System.in);

      Map<String, String> map = new HashMap<>();

      System.out.println("Map Operations:");

      System.out.println("1. Put Key-Value Pairs\n2. Remove Key\n3. Display Map\n4.
Exit");

      while (true) {

         System.out.print("Enter your choice: ");

         int choice = scanner.nextInt();

         scanner.nextLine();

         switch (choice) {

            case 1:

               System.out.print("Enter key-value pairs (key1=value1, key2=value2, ...): ");

               String[] pairs = scanner.nextLine().split(",");

               for (String pair : pairs) {

                  String[] keyValue = pair.split("=");

                  if (keyValue.length == 2) {

                     map.put(keyValue[0].trim(), keyValue[1].trim());

                  } else {

                     System.out.println("Invalid pair: " + pair);

                  }

               }

               break;

            case 2:

               System.out.print("Enter key to remove: ");

               String key = scanner.nextLine();
```

```
                map.remove(key);
                break;
            case 3:
                System.out.println("Map: " + map);
                break;
            case 4:
                return;
            default:
                System.out.println("Invalid choice.");}
        }
    }
}
```

```
C:\Users\HEMANTH KUMAR\OneDrive\Desktop\New folder>java map
Map Operations:
1. Put Key-Value Pairs
2. Remove Key
3. Display Map
4. Exit
Enter your choice: 1
Enter key-value pairs (key1=value1, key2=value2, ...): a=1,b=2
Enter your choice: 3
Map: {a=1, b=2}
Enter your choice: 2
Enter key to remove: a
Enter your choice: 3
Map: {b=2}
Enter your choice: 4
```

Week 2

(i)Perform setting up and Installing Hadoop in its three operating modes: Standalone, Pseudo distributed,

Fully distributed (ii)Use web based tools to monitor your Hadoop setup.REQUIRED TOOLS

1. Java JDK used to run the Hadoop since it's built using Java

2. 7Zip or WinRAR unzip Hadoop binary package, anything that unzips tar.gz

3. CMD or Powershell - used to test environment variables and run Hadoop

INSTALLATION STEPS

STEP-1:

Installation of Java JDK

Create a Java folder and install JDK in it. After installation, open up CMD and confirm Java is installed

>>java -version

Java is not an internal or external in this system.

STEP-2:

Configure Environment Variables for Java

Open Environment Variables and change user variables and system variables.

Add JAVA HOME with JDK bin path by creating new one

>>JAVA HOME-C:\java\jdk\bin

Add the JDK bin path to PATH in system variables and then click OK and close Environment Variables

>>java -version

java version "1.8.0 261"

STEP-3:

Installation of apache Hadoop

Download and install Hadoop-3.2.4 with tar.gz extension

STEP-4:

Installation of WinRAR

Download and install WinRAR and unzip the Hadoop-3.2.4 and open tar.gz file it will open in WinRAR and complete its process.

STEP-5:

Setup of JAVA HOME for Hadoop

Open etc folder in Hadoop folder and edit "hadoop-env.cmd" file

>> set JAVA_HOME=C:\Java\jdk-1.8

STEP-6:

Configure Hadoop environment variables

Open Environment variables and add new one in user variables.

>>HADOOP_HOME=C:\hadoop\bin

Now add Hadoop bin and sbin to PATH in system variables

>>C:\hadoop\bin >> C:\hadoop\sbin

Now check for Hadoop and its version in cmd

>> hadoop

>>Hadoop version → Hadoop 3.2.4

STEP-7:

Configure Hadoop

##core-site

Open hadoop etc>core-site.xml and click on edit. Add below code to <configuration> part and save it

Code:

```
<configuration>
<property>
<name>fs defaultFS</name>
<value>hdfs://localhost:9000</value>
<property>
</configuration>
```

##httpfs

Open hadoop etc httpfs-site.xml and click on edit. Add below code to <configuration> part and save it.

Now add a folder "data" in Hadoop folder In data folder add two more folders as

1) namenode

2) datanode

Code:

```
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.namenode.name.dir </name>
<value> c:\hadoop\data\namenode/value>
</property>
<property>
<name> dfs.datanode.data.dir</name>
<value> c:\hadoop\data\namenode </value>
</property>
</Configuration>
```


##mapreduce

Open hadoop se maprod-site xml and click on edit. Add below code to configuration part and save it.

Code:

```
<configuration>
<property>
<name>maproduce.framework.name</name>
<value>yarn</value>
<property>
</configuration>
```

##yarn

Code:

<configuration>

<property>

name>yarn nodemanager.aux-services</name>

<value> mapreduce_shuffle</value>

</property>

<property>

<name> yam.nodemanager.aux-services.mapreduce.shuffle.class</name>

<value>org.apache.hadoop mapred. ShuffleHandler/value>

</property>

</configuration>

Now change the bin of Hadoop with winutils bin from link.

STEP-8:

Installation of msvcp120.dll

Install msvcp120.dll and unzip it and open in WinRAR and now add msvcp120.dll to system32 folder in windows folder in C drive.

STEP-9:

Installation of Microsoft Visual C++

Download vc_redlist from Microsoft and complete its installation.

STEP-10:

Now in cmd,

>>hdfs namenode -format

>>cd/

>>cd hadoop

>>cd sbin

STEP-11:

Start HDFS daemons

>>start-dfs.cmd

Start VARN daemons

>>start-yam.cmd

STEP-12:

Use Web Portals

Open any browser and go for localhost:9870 to know whether your Hadoop is running

STEP-13:

Shutdown YARN and HDFS daemons

To shutdown YARN

>> stop-dfs.cmd

To shutdown HDFS

>> stop-yarn.cmd

To shutdown all cmd's

>> stop-all.cmd

Week 3

Implement the following file management tasks in Hadoop: Adding files and directories, Retrieving files,

Deleting files

Hint: A typical Hadoop workflow creates data files (such as log files) elsewhere and copies them into HDFS

using one of the above command line utilities.**Program:**

i)Adding files and directories mkdir –

To create a directory

```
C:\hadoop\hadoop-3.2.4\bin>hdfs dfs -mkdir /week3
```

touchz : It creates an empty file.

```
C:\hadoop\hadoop-3.2.4\bin>hdfs dfs -touchz /week3/filee1.txt
```

ii)Retrieving Files ls - This command is used to

list all the files.

Before creating directory

```
C:\hadoop\hadoop-3.2.4\bin>hdfs dfs -ls
ls: '.': No such file or directory
```

After creating directory and files

```
C:\hadoop\hadoop-3.2.4\bin>hdfs dfs -ls /
Found 1 items
drwxr-xr-x - admin2 supergroup          0 2025-01-13 9.30 /week3
```

iii)Deleting Files rm – This command deletes a file from

HDFS recursively.

```
C:\hadoop\hadoop-3.2.4\bin>hdfs dfs -rm/week3/file1.txt
Deleted /week3/file1.txt
```

Week 4

Run a basic Word Count MapReduce program to understand MapReduce Paradigm

Program:

```
import java.io.IOException;
import java.util.Iterator;
import java.util.StringTokenizer;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.TextOutputFormat;
public class WordCount {
public static class Map extends MapReduceBase implements
Mapper<LongWritable, Text, Text, IntWritable>
{
public void map (LongWritable key, Text value, OutputCollector <Text,
IntWritable> output, Reporter reporter) throws IOException
{
String line = value.toString();
StringTokenizer tokenizer = new StringTokenizer(line);
```

```
while (tokenizer.hasMoreTokens())
{value.set(tokenizer.nextToken());
output.collect(value, new IntWritable(1));}}}
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable>
{
public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException
{int sum = 0;
while (values.hasNext()) { sum += values.next().get();
}
output.collect(key, new IntWritable(sum));}
}
public static void main(String[] args) throws Exception
{JobConf conf = new JobConf(WordCount.class);
conf.setJobName("wordcount");
conf.setOutputKeyClass(Text.class);
conf.setOutputValueClass(IntWritable.class);
conf.setMapperClass(Map.class);
conf.setReducerClass(Reduce.class);
conf.setInputFormat(TextInputFormat.class);
conf.setOutputFormat(TextOutputFormat.class);
FileInputFormat.setInputPaths(conf, new Path(args[0]));
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
JobClient.runJob(conf);
}
}
```

Step 1: How to Save this File:

1.Create a Folder with the name WordCountApp

2.Save this file in the newly created folder as WordCount.java

Step 2: Prepare Input Data File:

1.Create a folder with the name input_data as sub folder of WordCountApp

2.Create a text file with the name input.txt in the folder input_data

3.Now write desired number of words to input.txt

Assume the following Words:

apple mango banana grapes banana apple orange mango grapes

Step 3: Copy this input.txt into HDFS

Command is

$ hdfs dfs -mkdir /WordCountApp

$ hdfs dfs -mkdir /WordCountApp/Input Next,

Change to folder WordCountApp

$hdfs dfs -copyFromLocal /input_data/input.txt' /WordCountApp/Input/

Step 4:

Create the classpath Command is

$export HADOOP_CLASSPATH =$(hadoop classpath)

Step 5: Compile the WordCount.java File

1.Create a folder with the name classes as a sub folder of WordCountApp

2.To Compile, command is:

javac -classpath ${HADOOP_CLASSPATH} -d '/home/hadoopusr/Desktop/WordCountApp/classes' '/home/hadoopusr/Desktop/WordCountApp/WordCount.java'

Source File: '/home/hadoopusr/Desktop/WordCountApp/WordCount.java'

Destination Folder for Classes:
'/home/hadoopusr/Desktop/WordCountApp/classes' On Successful Compilation, you have class files in classes folder

Step 6: Create a JAR File for the Application

1.Change to the folder where the class files are stored – classes

2.To Create a jar File, Command is:

$ jar -cvf WordCount.jar

3.To View jar file contents, Command is:

$ jar -tf WordCount.jar

Step 7: Now RUN the JAR File

Command is

$hadoop jar <jar file name><Class name><input url><output url>

$ hadoop jar WordCound.jar WordCount /WordCountApp/Input /WordCountApp/Output

Step 8: Viewing the Output

1.First list the files in the Output folder Command is

$ hdfs dfs -ls /WordCountApp/Output

Found 2 items

-rw-r--r-- 1 hadoopusr supergroup 0 2024-01-28 07:42 /WordCountApp/Output/_SUCCESS

-rw-r--r-- 1 hadoopusr supergroup 43 2024-01-28 07:42 /WordCountApp/Output/part-00000

2.Here, output file is found, then display it. Command is

$ hdfs dfs -cat /WordCountApp/Output/part-00000

Expected output:

Apple 2

banana 2

grapes 2

mango 2

orange 1

Week 5

a. Write a map reduce program that mines weather data. Weather sensors collecting data every hour at

many locations across the globe gather a large volume of log data, which is a good candidate for analysis

with Map Reduce, since it is semi structured and record-oriented.

b. Use MapReduce to find the shortest path between two people in a social graph. Hint: Use an adjacencylist to model a graph, and for each node store the distance from the original node, as well as a back

pointer to the original node. Use the mappers to propagate the distance to the original node, and the

reducer to restore the state of the graph. Iterate until the target node has been reached.

Program:

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
public class MyMaxMin {
public static class MaxTemperatureMapper extends Mapper<LongWritable, Text, Text, Text>
{
```

```
public void map(LongWritable arg0, Text Value, Context context)throws IOException,
InterruptedException
{
String line = Value.toString(); if (!(line.length() == 0)){
String date = line.substring(6, 14);  //date
float temp_Max = Float.parseFloat(line.substring(39, 45).trim());
float temp_Min = Float.parseFloat(line.substring(47, 53).trim());
if (temp_Max > 35.0){
context.write(new Text("Hot Day " + date), new Text(String.valueOf(temp_Max)));
}
if (temp_Min < 10)
{
context.write(new Text("Cold Day " + date),new Text(String.valueOf(temp_Min)));
}}}}
public static class MaxTemperatureReducer extends Reducer<Text, Text, Text, Text>
{
public void reduce(Text Key, Iterator<Text> Values, Context context)throws IOException,
InterruptedException
{
String temperature = Values.next().toString();
context.write(Key, new Text(temperature));
}}
public static void main(String[] args) throws Exception
{
Configuration conf = new Configuration();
Job job = new Job(conf, "weather example");
job.setJarByClass(MyMaxMin.class); //Assigning the driver class name
job.setMapOutputKeyClass(Text.class); //Key type coming out of mapper
job.setMapOutputValueClass(Text.class); //value type coming out of mapper
job.setMapperClass(MaxTemperatureMapper.class);
job.setReducerClass(MaxTemperatureReducer.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path OutputPath = new Path(args[1]);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
OutputPath.getFileSystem(conf).delete(OutputPath);
System.exit(job.waitForCompletion(true) ? 0 : 1);}
}
```

Step 1: How to Save this File:

1. Create a Folder with the name **WeatherApp**
2. Save this file in the newly created folder as **MyMaxMin.java**

Step 2: Prepare Input Data File:

1. Create a folder with the name **input_data** as sub folder of **WeatherApp**

2. Create a text file with the name **Weather.txt** in the folder **input_data**

Step 3: Copy this Weather.txt into HDFS

Command is

$ hdfs dfs -mkdir /WeatherApp

$ hdfs dfs -mkdir /WeatherApp/Input Next, Change to folder WeatherApp
$hdfs dfs -copyFromLocal /input_data/Weather.txt' /WeatherApp/Input/

Step 4: Create the classpath
Command is **$export HADOOP_CLASSPATH =$(hadoop classpath)**
Step 5: Compile the MyMaxMin.java File
1. Create a folder with the name **classes** as a sub folder of **WeatherApp**
2. To Compile, command is:

javac -classpath ${HADOOP_CLASSPATH} -d '/home/hadoopusr/Desktop/WeatherApp/classes' '/home/hadoopusr/Desktop/WeatherApp/MyMaxMin.java'
Source File: '/home/hadoopusr/Desktop/WeatherApp/WordCount.java'
Destination Folder for Classes: '/home/hadoopusr/Desktop/WeatherApp/classes' On Successful Compilation, you have class files in **classes** folder

Step 6: Create a JAR File for the Application
1. Change to the folder where the class files are stored – **classes**
2. To Create a jar File, Command is:

$ jar -cvf Weather.jar .

3. To View jar file contents, Command is:

$ jar -tf Weather.jar

Step 7: Now RUN the JAR File
Command is
$hadoop jar <jar file name><Class name><input url><output url>

$ hadoop jar Weather.jar MyMaxMin /WeatherApp/Input /WeatherApp/Output
Step 8: Viewing the Output
First list the files in the Output folder Command is

```
C:\hadoop\hadoop-3.2.4\bin>$ hdfs dfs -ls /WeatherApp/Output
Found 2 items
-rw-r--r--   1 hadoopusr supergroup    0    2024-01-28 10:06 /WeatherApp/Output/_SUCCESS
-rw-r--r--   1 hadoopusr supergroup    8489 2024-01-28 10:06 /WeatherApp/Output/part-r-00000
>$ hdfs dfs -cat /WeeatherApp/Output/part-r-00000
```

Expected Output:

```
Cold Day 20150101        -21.8
Cold Day 20150102        -24.9
Cold Day 20150103        -28.2
Cold Day 20150104        -28.9
Cold Day 20150105        -29.3
Cold Day 20150106        -26.3
Cold Day 20150107        -28.7
Cold Day 20150108        -24.1
Cold Day 20150109        -20.3
Cold Day 20150110        -25.8
Cold Day 20150111        -28.2
Cold Day 20150112        -29.1
Cold Day 20150113        -29.9
Cold Day 20150114        -29.0
Cold Day 20150115        -24.2
Cold Day 20150116        -24.6
Cold Day 20150117        -23.2
Cold Day 20150118        -23.0
Cold Day 20150119        -30.4
Cold Day 20150120        -24.7
Cold Day 20150121        -24.1
Cold Day 20150122        -27.5
Cold Day 20150123        -29.3
Cold Day 20150124        -30.3
```

Week 6

a. Hive Queries a) Install Hive Framework b) Implement Hive to create, alter, and drop databases, tables

b. Hive Queries Implement hive queries and joins to perform display and retrieve the data

Process:

a) Hive Queries

1) Install Hive Framework:

To install Apache Hive, you typically need a Hadoop environment. Below are the steps to install Hive:

Step-by-Step Hive Installation:

1. **Download Hive:** Download the latest version of Apache Hive from the official website: Apache Hive Download.

2. **Extract Hive**: After downloading the .tar.gz file, extract it using:

$ tar -xvzf hive-x.x.x.tar.gz

3. **Set Environment Variables**: Edit your .bashrc or .bash_profile file to include Hive environment variables:

$ export HIVE_HOME=/path/to/hive

$ export PATH=$PATH:$HIVE_HOME/bin

$ export HADOOP_HOME=/path/to/hadoop

$ export PATH=$PATH:$HADOOP_HOME/bin

4. **Configure Hive:** Set up the configuration files in the conf directory (e.g., hive-site.xml, hadoop-env.sh).

o **hive-site.xml**: Contains essential Hive configurations like Metastore URIs, warehouse location, etc.

o **hadoop-env.sh**: Set Hadoop environment variables like JAVA_HOME, etc.

5. **Start Hadoop Services:** Ensure that Hadoop services are running, especially HDFS and YARN (or MR1 if using older versions).

6. **Start Hive Server:** You can start the Hive shell by running:

$ hive

2) Implement Hive to Create, Alter, and Drop Databases, Tables:

In Hive, you can manage databases and tables using the following commands:

1. Create a Database:

CREATE DATABASE IF NOT EXISTS my_database;

2. Use the Database: Switch to the created database:

USE my_database;

3. Create a Table: Example of creating a simple table:

CREATE TABLE IF NOT EXISTS employees (

id INT, name STRING,

department STRING,

salary FLOAT)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;

4. Alter a Table:

o To add a new column:

ALTER TABLE employees ADD COLUMNS (hire_date STRING);

o To change the name of a column:

ALTER TABLE employees CHANGE COLUMN name employee_name STRING;

5. Drop a Table:

DROP TABLE IF EXISTS employees;

DROP DATABASE IF EXISTS my_database CASCADE;

b) Hive Queries

1. **Select All Data from a Table:**

SELECT * FROM employees;

2. **Select Specific Columns:**

SELECT name, salary FROM employees;

3. **Filter Data Using WHERE Clause:**

SELECT * FROM employees WHERE department = 'Sales';

4. **Sort Data Using ORDER BY:**

SELECT * FROM employees ORDER BY salary DESC;

5. **Join Tables:** You can join multiple tables in Hive using JOIN clauses. Example of an INNER JOIN:

SELECT e.name, d.department_name FROM employees e

JOIN departments d

ON e.department = d.department_id;

Example of a LEFT JOIN:

SELECT e.name, d.department_name FROM employees e

LEFT JOIN departments d

ON e.department = d.department_id;

6. **Group By and Aggregation:** Use GROUP BY to group data and perform aggregation functions like COUNT, SUM, etc.

SELECT department, AVG(salary) FROM employees GROUP BY department;

7. **Limit Results:** To limit the number of results returned:

SELECT * FROM employees LIMIT 10;

8. **Insert Data into Table:** To insert data into a Hive table from another table or a file. Example to insert data:

INSERT INTO TABLE employees VALUES (101, 'John Doe', 'IT', 75000);

You can also load data from a file:

LOAD DATA LOCAL INPATH '/path/to/employees.csv' INTO TABLE employees;

Week 7

Demonstrate Spark SQL on Hive

a. Create a SQLContext object and load the Parquet file into dataFrame

b. Load the Dataframe into Hive table.

c. Verify the created Hive table in Hive environment

d. Execute Spark SQL query.

Process:

1) Create a SQLContext Object and Load the Parquet File into DataFrame:

To create a SQLContext and load the Parquet file into a DataFrame in Spark:

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \

.appName("Spark SQL with Hive Example") \

.enableHiveSupport() \ # Enabling Hive support

.getOrCreate()

df = spark.read.parquet("path")

df.show()
```

2) Load the DataFrame into a Hive Table

To load the data into a Hive table, you can either register the DataFrame as a temporary view or directly write it into a Hive table.

Option 1: Register DataFrame as a Temporary View:

```
# Register DataFrame as a temporary view in Spark SQL

df.createOrReplaceTempView("temp_view_parquet_data")
```

Option 2: Write DataFrame into a Hive Table:

```
# Save DataFrame into Hive table (if the table doesn't exist)

df.write.mode("overwrite").saveAsTable("hive_table_name")
```

This will create a Hive table called hive_table_name and load the data from the DataFrame into it. If the table already exists, it will overwrite it.

3) Verify the Created Hive Table in Hive Environment:

To verify the Hive table is created and the data is stored in it, you can run a Spark SQL query to list the tables in the Hive database:

```
# Execute a SQL query to list all tables in the current Hive database

spark.sql("SHOW TABLES").show()

# Verify the content of the table you created

spark.sql("SELECT * FROM hive_table_name").show()
```

4) Execute a Spark SQL Query:

Now, you can execute a Spark SQL query on the hive_table_name to interact with the data in Hive:

```
# Example: Execute Spark SQL query

query_result = spark.sql("SELECT column_name FROM hive_table_name WHERE some_condition")

# Show the query result

query_result.show()
```


Week -8

A Sales Analyst need to analyze 100 million historical sales data stored on Hadoop Data Lake in order to find

out their best selling products, most frequently purchasing customers, maximum revenue generated by a product

and customer. This analysis would help them to provide offers to customers, find out the best selling product

partners. Below is the sample of their sale dataset named as SalesData.csv Schema –

cust_id, cust_name, cust_email, date, prod_id,

prod_name,prod_price103,john, Bellevue

102,james, Renton

101, jayveer, Seattle

104, Meena, Renton

105,Marry, Bellevue

Below are the analysis requirements to create a dataset:
Below requirements are related to DataFrame creation

and working with different file formats category.

Create a Data Frame from the data and write Spark SQL query to compute the average sale of every customer.

Store the output as a parquet file.

```
from pyspark.sql import SparkSession

from pyspark.sql.functions import avg, col

# Initialize Spark Session

spark = SparkSession.builder.appName("SalesAnalysis").getOrCreate()

# Read SalesData.csv from Hadoop Data Lake

sales_df = spark.read.option("header", "true").csv("hdfs://path_to_hadoop/SalesData.csv")

# Convert product price column to float

sales_df = sales_df.withColumn("prod_price", col("prod_price").cast("float"))

# Compute the average sale per customer

avg_sales_df = sales_df.groupBy("cust_id",

"cust_name").agg(avg("prod_price").alias("avg_sale"))

# Store the output as a Parquet file

avg_sales_df.write.mode("overwrite").parquet("hdfs://path_to_hadoop/output/avg_sales.parq

uet")

# Show the result

avg_sales_df.show()
```

Output:

```
+-------+--------+---------+
|cust_id|cust_name|avg_sale|
+-------+--------+---------+
|    101| Jayveer|     75.0|
|    102|   James|    175.0|
|    103|    John|    275.0|
|    104|   Meena|    400.0|
|    105|   Marry|    500.0|
+-------+--------+---------+
```

2. Store the Output as a Parquet File

Now, save the DataFrame as a Parquet file for optimized storage.

CODE:

avg_sales_df.write.mode("overwrite").parquet("hdfs://path_to_hadoop/output/avg_sales.parquet")

Week 9

SPARK SQL query a.) Create a Data Frame from the data and write Spark SQL query to compute and findthe

most sold product. Store the output as JSON file. b.) Create two DataFrames from the two datasets. Write

Spark SQL query to join both and compute - The number of transactions made by customers at "Bellevue" -

Compute the total amount of transactions carried by every city

Process:

1. Import Required Libraries CODE:

from pyspark.sql import SparkSession from

pyspark.sql.functions import col, sum, count

Initialize Spark Session

spark = SparkSession.builder.appName("SparkSQLQueries").getOrCreate()

2. Create DataFrames

Assume we have two datasets:

1.sales_data.csv: Contains sales details

2.transactions_data.csv: Contains transaction details **Load and**

Create DataFrames CODE:

Load datasets

sales_df = spark.read.csv("sales_data.csv", header=True, inferSchema=True) transactions_df

= spark.read.csv("transactions_data.csv", header=True, inferSchema=True)

Register DataFrames as SQL tables sales_df.createOrReplaceTempView("sales")

transactions_df.createOrReplaceTempView("transactions")

3. Find the Most Sold Product

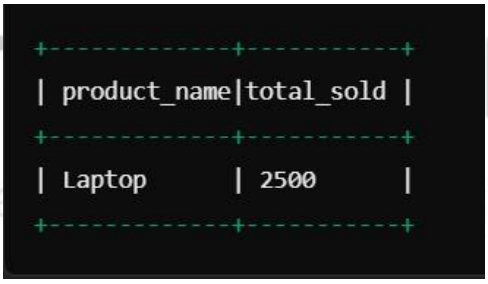
CODE:

most_sold_product = spark.sql("""

```
    SELECT product_name, SUM(quantity) as total_sold
    FROM sales
    GROUP BY product_name
    ORDER BY total_sold DESC
    LIMIT 1
""")
# Store output as JSON
most_sold_product.write.mode("overwrite").json("most_sold_product.json")
# Show the result
most_sold_product.show()
```

Output:

Most sold product



```
+------------+-----------+
| product_name|total_sold |
+------------+-----------+
| Laptop      | 2500      |
+------------+-----------+
```

4. Join DataFrames & Compute Required Queries CODE:

```
# Join sales and transactions DataFrames joined_df
= spark.sql("""
    SELECT t.transaction_id, t.customer_id, t.city, s.product_name, s.quantity, s.amount
    FROM transactions t
    JOIN sales s ON t.transaction_id = s.transaction_id
""")
# Register joined DataFrame as a table joined_df.createOrReplaceTempView("joined_sales")
# Compute number of transactions in Bellevue
bellevue_transactions = spark.sql("""
    SELECT COUNT(*) as total_transactions
    FROM joined_sales
```

```
    WHERE city = 'Bellevue'
""")
bellevue_transactions.show()
```

Output:

Number of Transactions in Bellevue

```
+------------------+
| total_transactions|
+------------------+
|  1350            |
+------------------+
```

```
# Compute total transaction amount per city total_amount_per_city
= spark.sql("""
    SELECT city, SUM(amount) as total_amount
    FROM joined_sales
    GROUP BY city
    ORDER BY total_amount DESC
""")
total_amount_per_city.show()
```

Output:

Total Transaction Amount by City

```
+------------+--------------+
| city       | total_amount |
+------------+--------------+
| New York   | 5000000      |
| Seattle    | 3200000      |
| Bellevue   | 2500000      |
| Austin     | 1900000      |
+------------+--------------+
```

Week 10

Char Count - Find and display the number of occurrences of each character in a text file- Spark. 11. Performthe

filter, count, distinct, map, flatMap RDD Operations in Spark.

Program:

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("CharCountRDD").getOrCreate()
# Create SparkContext sc =
spark.sparkContext
# Read the text file from Hadoop
rdd = sc.textFile("hdfs://path_to_hadoop/input_text.txt")
# Convert each line into characters and flatten the structure
char_rdd = rdd.flatMap(lambda line: list(line))
# Map each character to a (character, 1) tuple
char_count_rdd = char_rdd.map(lambda char: (char, 1))
# Reduce by key to get the total count of each character
char_count_result = char_count_rdd.reduceByKey(lambda a, b: a + b)
# Collect and display the results
char_count_output = char_count_result.collect()
print("Character Count Results:")
for char, count in char_count_output:
    print(f"'{char}': {count}") #
Save the output in Hadoop
char_count_result.saveAsTextFile("hdfs://path_to_hadoop/output/char_count")
# Stop Spark session spark.stop()
```

Sample Input (input_text.txt):

Hello World

Spark is awesome!

Expected Output:

Character Count Results:

'H': 1

'e': 2

'l': 3

'o': 2

' ': 3

'W': 1

'r': 1

'd': 1

'S': 1

'p': 1 'a':

2 'r': 1

'k': 1

'i': 1

's': 2

'w': 1

'o': 1

'm': 1

'!': 1

Week 11

Perform the

filter, count,

distinct, map,

flatMap

RDDOperations

inSpark.

Program:

```
from pyspark.sql import SparkSession
from pyspark import SparkContext #
Initialize Spark Session
spark = SparkSession.builder.appName("RDDOperations").getOrCreate() sc
= spark.sparkContext  # Get the SparkContext

# Sample Input Data (Simulating a Text File) sample_text
= [
    "Hello Spark",
    "Spark is powerful",
    "Big Data with Spark",
    "Apache Spark is amazing"
]
# Create an RDD from the sample text text_rdd
= sc.parallelize(sample_text)
```

1.filter() - Keep lines containing "Spark"

filtered_rdd = text_rdd.filter(lambda line: "Spark" in

line) print("\n Filtered Lines (Containing 'Spark'):")

print(filtered_rdd.collect())

1. count() - Count total number of lines

line_count = text_rdd.count() print(f"\n Total number

of lines: {line_count}") **3. distinct() - Get unique**

words from the text

distinct_words_rdd = text_rdd.flatMap(lambda line: line.split(" ")).distinct()

print("\n Distinct Words:") print(distinct_words_rdd.collect())

4. map() - Convert each line to its length line_length_rdd

= text_rdd.map(lambda line: len(line)) print("\n Line

Lengths:") print(line_length_rdd.collect())

5. flatMap() - Split lines into words

words_rdd = text_rdd.flatMap(lambda line: line.split(" "))

print("\n Words List:") print(words_rdd.collect()) **# Stop**

Spark Session spark.stop() **O/P:**

Filtered Lines (Containing 'Spark'):

['Hello Spark', 'Spark is powerful', 'Big Data with Spark', 'Apache Spark is amazing']

Total number of lines: 4

Distinct Words:

['Hello', 'Spark', 'is', 'powerful', 'Big', 'Data', 'with', 'Apache', 'amazing']

Line Lengths:

[12, 18, 19, 24]

Words List:

['Hello', 'Spark', 'Spark', 'is', 'powerful', 'Big', 'Data', 'with', 'Spark', 'Apache', 'Spark', 'is', 'amazing']

Week 12

Spark joins: Consider a scenario where 2 datasets of a leading retail client to be joined with one another using

Spark joins. Customer dataset: Sales dataset: Schema Details: 101 ravi 1 102 keerth 2 101 Syam 1 101 Geetha 1

103 Dawn 3 101 ravi 1 102 keerth 2 101 Syam 1 101 Geetha 1 103 Dawn 3 AR20 Computer Science andEngineering Aditya Engineering College (A) 66 Customer schema (customer id,customer name,product id)

Sales Schema (product id,product name and price) Join both datasets with common key Product id and print

customer id, customer name, product name and price.

Program:

```
from pyspark.sql import SparkSession
# Initialize Spark Session
spark = SparkSession.builder.appName("RetailDataJoin").getOrCreate() #
Create Customer Dataset (customer_id, customer_name, product_id)
customer_data = [(101, "Ravi", 1),
                 (102, "Keerth", 2),
                 (101, "Syam", 1),
                 (101, "Geetha", 1),
                 (103, "Dawn", 3)]
customer_schema = ["customer_id", "customer_name", "product_id"] customer_df
= spark.createDataFrame(customer_data, customer_schema)
# Create Sales Dataset (product_id, product_name, price)
sales_data = [(1, "Laptop", 50000),
              (2, "Mobile", 20000),
              (3, "Tablet", 15000)]
sales_schema = ["product_id", "product_name", "price"] sales_df
= spark.createDataFrame(sales_data, sales_schema)
# Perform Inner Join on product_id
```

```
joined_df = customer_df.join(sales_df, "product_id")
# Select required columns
result_df = joined_df.select("customer_id", "customer_name", "product_name", "price")
# Show the result
result_df.show() #
Stop Spark session
spark.stop()
```

Output:

```
+-----------+-------------+-----------+------+
|customer_id|customer_name|product_name|price|
+-----------+-------------+-----------+------+
|        101|         Ravi |     Laptop |50000|
|        101|        Syam  |     Laptop |50000|
|        101|       Geetha |     Laptop |50000|
|        102|       Keerth |     Mobile |20000|
|        103|        Dawn  |     Tablet |15000|
+-----------+-------------+-----------+------+
```