# Table of Contents

**EX: 1** **DATE:19/03/2024**

# EXPLORATION OF THE HADOOP INSTALLATION

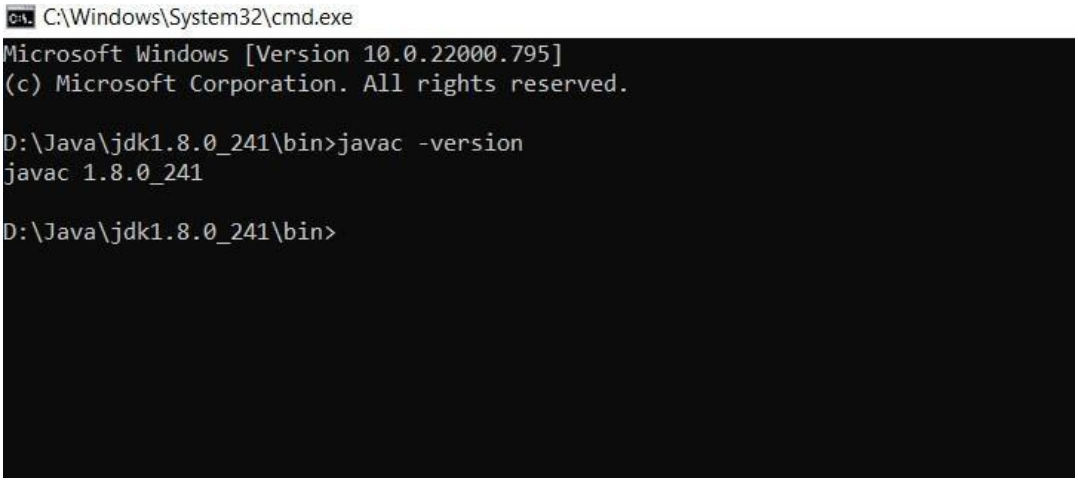
## AIM:

To create an Exploration of the hadoop environment.

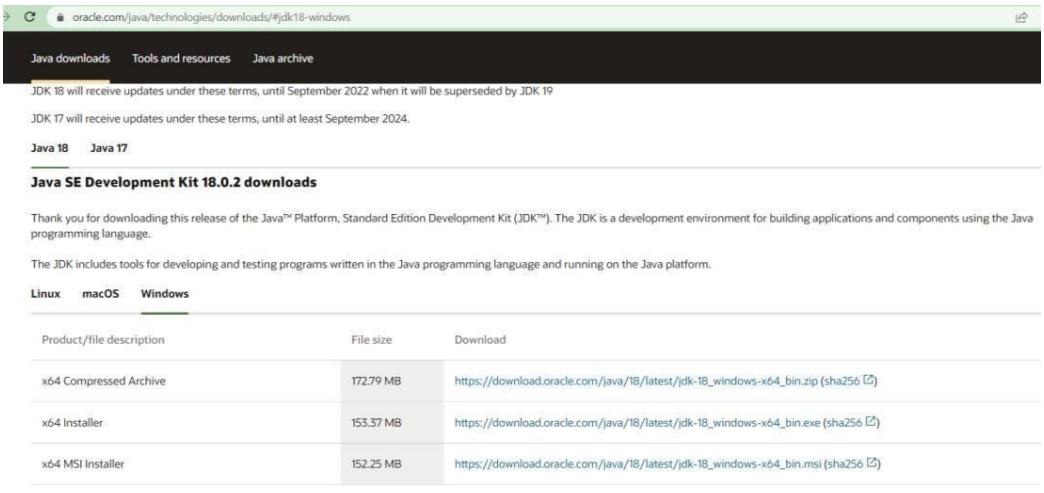
## ALGORITHM:

**STEP1:** To install Hadoop, First you should have Java version 1.8 in your system.

**STEP 2:** Check your java version in this command on command prompt **javac –version**




**STEP3:** If java is not installed in your system



**STEP4:** Download the file according to your operating system. Keep the java folderdirectly under the local disk directory

**(D:\Java\jdk1.8.0_241\bin)**

**STEP5:** After downloading java version 1.8, download hadoop version 3.1

Hadoop Link:
https://archive.apache.org/dist/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz

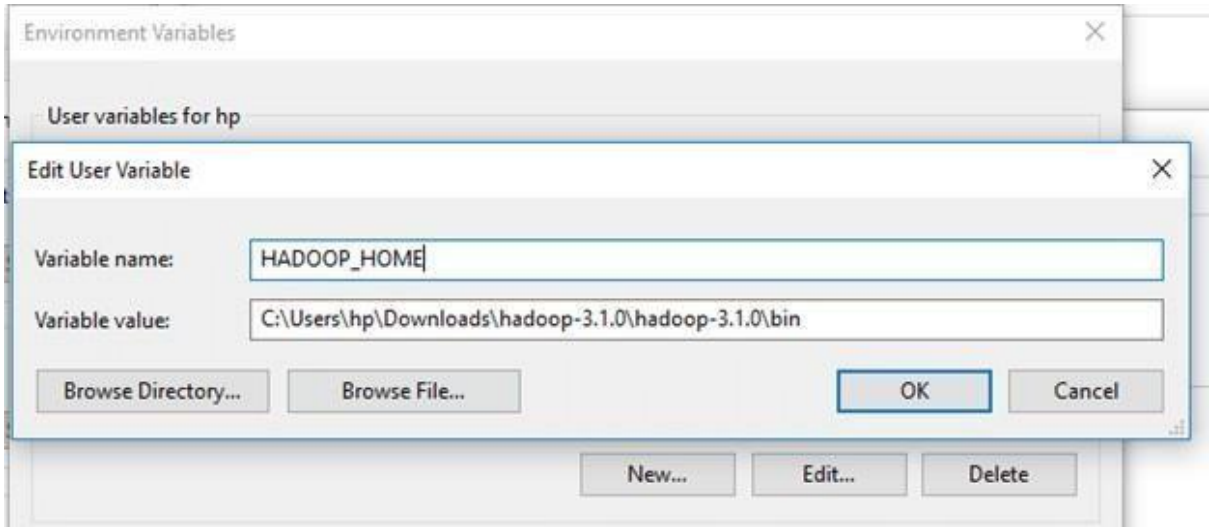
**STEP6:** Extract it to a folder

## SETUP SYSTEM ENVIRONMENT VARIABLES

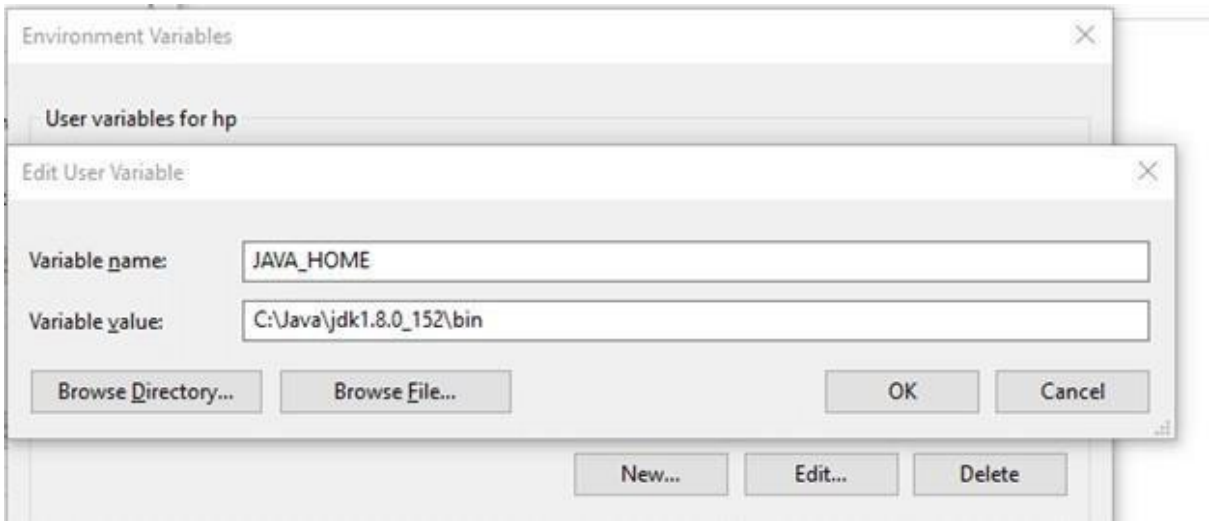
**STEP7:** Open control panel to edit the system environment variable

**STEP8:** Go to environment variable in system properties

**STEP9:** Create a new user variable. Put the Variable name as HADOOP_HOME and Variable value as the path of the bin folder where you extracted hadoop

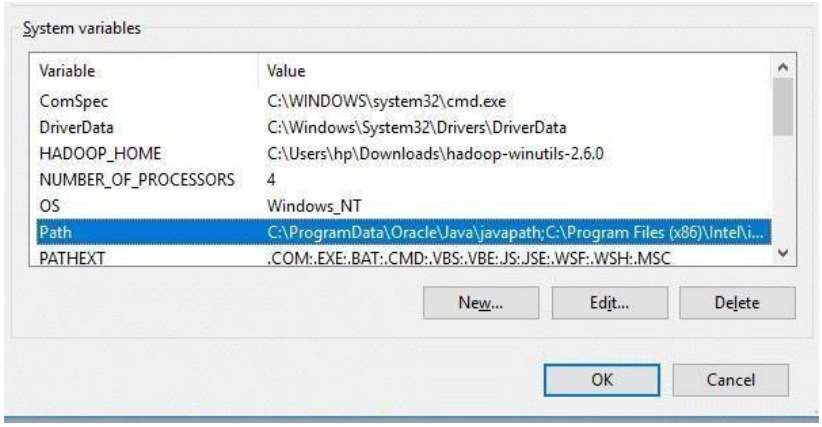


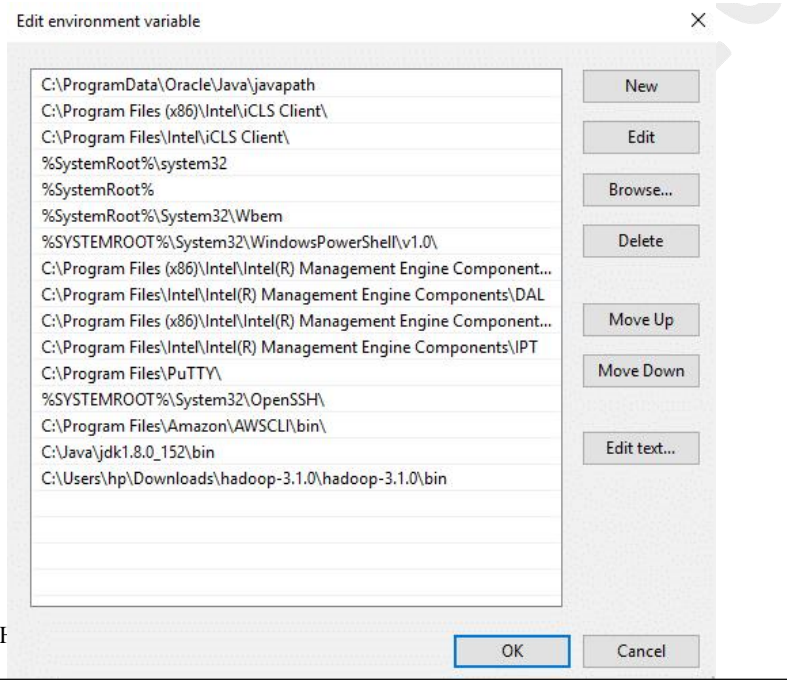
**STEP10:** create a new user variable with variable name as JAVA_HOME and variable value as the path of the bin folder in the Java directory.




**STEP11:** Now we need to set Hadoop bin directory and Java bin directory path in system variable path.

**STEP12:**Edit Path in system variable



**RESULT:**

Thus the Program has been successfully completed.

**EXP:2** **DATA:26/03/2024**

# Hadoop Implementation of file management tasks, such as Adding files and directories, Retrieving files and Deleting files

## AIM:

To Hadoop Implementation of file management tasks, such as Adding files and directories, Retrieving files and Deleting files

## ALGORITHM:

### 1. Create a directory in HDFS at given path(s).

**SYNTAX :-**

hadoop fs -mkdir <paths>

**Example:**

hadoop fs -mkdir /user/saurzcode/dir1 /user/saurzcode/dir2

### 2. List the contents of a directory.

**SYNTAX :-**

hadoop fs -ls <args>

**Example:-**

hadoop fs -ls /user/saurzcode

### 3. Upload and download a file in HDFS.

**SYNTAX :-**

hadoop fs -put:

Copy single src file, or multiple src files from local file system to the Hadoop data file system

**SYNTAX:-**

hadoop fs -put <localsrc> ... <HDFS_dest_Path>

**Example:**

hadoop fs -put /home/saurzcode/Samplefile.txt /user/

saurzcode/dir3/**Download:hadoop fs -get:**

Copies/Downloads files to the local file system

**SYNTAX:-**

hadoop fs -get <hdfs_src> <localdst>

**Example:**

hadoop fs -get /user/saurzcode/dir3/Samplefile.txt /home/

## 4. See contents of a file

Same as unix cat command:

**SYNTAX:-**

hadoop fs -cat <path[filename]>

**Example:**

hadoop fs -cat /user/saurzcode/dir1/abc.txt

## 5. Copy a file from source to destination

This command allows multiple sources as well in which case the destination must be a directory.

**SYNTAX:-**

hadoop fs -cp <source> <dest>

**Example:**

hadoop fs -cp /user/saurzcode/dir1/abc.txt /user/saurzcode/ dir2

## 6. Copy a file from/To Local file system to HDFS CopyFromLocal

**SYNTAX:-**

hadoop fs -copyFromLocal <localsrc> URI

**Example:**

hadoop fs -copyFromLocal /home/saurzcode/abc.txt  /user/ saurzcode/abc.txt

Similar to put command, except that the source is restricted to a local file reference.

**copyToL**

**ocal**

**SYNTAX:**

hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

Similar to get command, except that the destination is restricted to a local file reference.

## 7. • Move file from source to destination.

Note:- Moving files across filesystem is not permitted.

**SYNTAX:-**

hadoop fs -mv <src> <dest>

**Example:**

hadoop fs -mv /user/saurzcode/dir1/abc.txt /user/saurzcode/ dir2

## 8. • Remove a file or directory in HDFS.

Remove files specified as argument. Deletes directory only when it is empty

**SYNTAX:-**

hadoop fs -rm <arg>

**Example:**

hadoop fs -rm /user/saurzcode/dir1/abc.txt **Recursive version of delete.**

**SYNTAX:-**
hadoop fs -rmr <arg>

**Example:**

hadoop fs -rmr /user/saurzcode/

## 9. Display last few lines of a file.Similar to tail command in Unix.

**SYNTAX:-**

hadoop fs -tail <path[filename]>

**Example:**

hadoop fs -tail /user/saurzcode/dir1/abc.txt

## 10. Display the aggregate length of a file.

**SYNTAX:-**

hadoop fs-du <path>

**Example:**

hadoop fs -du /user/saurzcode/dir1/abc.txt.

**RESULT :-**

Thus the Program Implementation of file management tasks has been successfully completed

**EX.NO:- 3** **DATE:-13/04/2024**

# Implement of Matrix Multiplication with Hadoop Map Reduce

## AIM :-

To implement of Matrix Multiplication with Hadoop Map

## ALGORITHM:

1. Prepare Data: Format matrices \(A\) and \(B\) as key-value pairs for Hadoop.
2. Mapper: Emit \((k, (i, A[i][k]))\) for matrix \(A\) and \((k, (j, B[k][j]))\) for matrix \(B\).
3. Shuffle and Sort: Group key-value pairs by key \(k\).
4. Reducer: Multiply pairs \((i, A[i][k])\) and \((j, B[k][j])\), and sum partial results for each \((i, j)\).
5. Output: Emit final matrix \(C\) elements \((i, j)\) as the sum of partial products.

## Reduce Program :-

```
import java.io.IOException; import java.util.;
import
java.util.AbstractMap.SimpleEntry;
import java.util.Map.Entry;
import
org.apache.hadoop.fs.Path;
importorg.apache.hadoop.conf.;
import org.apache.hadoop.io.;
import org.apache.hadoop.mapreduce.;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
importorg.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

 public class TwoStepMatrixMultiplication {

    public static class Map extends Mapper<LongWritable, Text, Text, Text>
       { public void map(LongWritable key, Text value, Context
 context) throws IOException,
          InterruptedException { String line =
          value.toString();
          String[] indicesAndValue = line.split(",");
          Text outputKey = new Text();
          Text outputValue = new Text();
          if(indicesAndValue[0].equals("A"))
          {
           outputKey.set(indicesAndValue[2]);
  outputValue.set("A," + indicesAndValue[1] +
  ","
```

```
    +indicesAndValue[3]);
        context.write(outputKey,
        outputValue);
            }
            else
             {
               outputKey.set(indicesAndValue[1]);
               outputValue.set("B," + indicesAndValue[2] +
               "," +
     indicesAndValue[3]);
               context.write(outputKey,  outputValue);
            }
public static class Reduce extends Reducer<Text, Text, Text, Text> {
          public void reduce(Text key, Iterable<Text> values, Context context) throws IOException,
     InterruptedException {
          String[] value;

     ArrayList<Entry<Integer, Float>> listA = newArrayList<Entry<Integer,
     Float>>();
     ArrayList<Entry<Integer, Float>> listB = new
     ArrayList<Entry<Integer, Float>>();
     for (Text val : values)
     {
     value = val.toString().split(",");
    if (value[0].equals("A"))
     {
     listA.add(new SimpleEntry<Integer, Float>(Integer.parseInt(value[1]),
     Float.parseFloat(value[2])));
               }
                else
                {
     listB.add(new SimpleEntry<Integer, Float>(Integer.parseInt(value[1]),
     Float.parseFloat(value[2])));
               }
            }
            Stri
     ng i; float
     a_ij; String
     k;
     float b_jk;
    Text outputValue = new
    Text(); for (Entry<Integer,
    Float> a: listA)
     {
     i =
     Integer.toString(a.getKey());
     a_ij = a.getValue();
               for (Entry<Integer, Float> b :
               listB) { k
               =Integer.toString(b.getKey());
     b_jk = b.getValue();
```

```
    outputValue.set(i + "," + k + "," + Float.toString(a_ijb_jk));
   context.write(null, outputValue);
                }
             }
          }
       }


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

          Job job = new Job(conf,
    "MatrixMatrixMultiplicationTwoSteps");
    job.setJarByClass(TwoStepMatrixMultiplication.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(Text.class); job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
FileInputFormat.addInputPath(job, new Path("hdfs://
    127.0.0.1:9000/matrixin"));
                FileOutputFormat.setOutputPath(job, new Path("hdfs://
    127.0.0.1:9000/matrixout"));
   job.waitForCompletion(true);
       }
```

**RESULT :-**

Thus the Program Implementation of file management tasks has been successfully completed

**EX.NO:- 4** **DATE:-23/04/2024**

# Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

**AIM:-** TO Run a basic Word Count Map Reduce program to understand Map Reduce Paradigm.

## ALGORITHM:

1. Mapper: Read each line of the input text, split it into words, and emit each word with a count of 1.

2. Shuffle and Sort: Hadoop groups the emitted word-count pairs by word.

3. Reducer: Sum the counts for each word to get the total occurrences of each word.

4. Output: Emit each word with its total count.

5. Result: The final output is a list of words with their respective counts.

## PROGRAM :-

```
import java.io.IOException;
 import
 java.util.StringTokenizer;
 import
org.apache.hadoop.io.IntWritable;
import
org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import
org.apache.hadoop.mapreduce.Mapper;
import
org.apache.hadoop.mapreduce.Reducer;
import
org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import
```

```
org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import
org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;
 public class WordCount
{
public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>
{
 public void map(LongWritable key, Text value,Context context) throws
IOException,InterruptedException{
String line = value.toString();
StringTokenizer tokenizer = newStringTokenizer(line);
 while (tokenizer.hasMoreTokens())
 {
 value.set(tokenizer.nextToken())
context.write(value, new IntWritable(1));
}
}
 }
public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable>
{
 public voidreduce(Text key, Iterable<IntWritable> values,Context context) throws
IOException,InterruptedException { int sum=0;
for(IntWritable x: values)
{
sum+=x.get();
}
 context.write(key, new IntWritable(sum));
}
}
  public static void main(String[] args) throws Exception
{ Configuration conf= new Configuration();
 Job job = newJob(conf,"My Word Count Program");
job.setJarByClass(WordCount.class);
```

```
job.setMapperClass(Map.class);
job.setReducerClass(Reduce.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(IntWritable.class);
job.setInputFormatClass(TextInputFormat.class);
job.setOutputFormatClass(TextOutputFormat.class);
Path outputPath = new Path(args[1])//Configuring the
input/output path from the filesystem into the job
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
//deleting the output path automatically from hdfs so that we don't have to delete it explicitly
outputPath.getFileSystem(conf).delete(outputPath); //exiting the job only if the flag value
becomes false
System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

**RESULT :-**

Thus, the Program has been successfully completed

**EX.NO:- 5** **DATE:-14/05/2024**

# Implementation of K-means clustering using Map Reduce

**Aim:-** To Implementation of K-means clustering using Map Reduce.

## ALGORITHM:-

1. Initialize: Randomly select initial cluster centroids.
2. Mapper: Assign each data point to the nearest centroid, emitting the centroid and data point as key-value pairs.
3. Shuffle and Sort: Group data points by assigned centroids.
4. Reducer: Compute the new centroids by averaging the data points in each group.
5. Iterate: Repeat the Mapper and Reducer steps until the centroids stabilize.

## PROGRAM :-

```
package it.unipi.hadoop.mapreduce;

import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import it.unipi.hadoop.model.Point;

import java.io.IOException;

public class KMeansMapper extends Mapper<LongWritable, Text, IntWritable, Point> {
	private Point[] centroids;
	private int p;
	private final Point point = new Point();
	private final IntWritable centroid = new IntWritable();

	@Override
	public void setup(Context context) {
		int k = Integer.parseInt(context.getConfiguration().get("k"));
		this.p = Integer.parseInt(context.getConfiguration().get("distance"));
		this.centroids = new Point[k];
		for (int i = 0; i < k; i++) {
			String[] centroid = context.getConfiguration().getStrings("centroid." + i);
			this.centroids[i] = new Point(centroid);
		}
	}
```

```
@Override
	public void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
		// Construct the point
		String[] pointString = value.toString().split(",");
		point.set(pointString);

		// Initialize variables
		float minDist = Float.POSITIVE_INFINITY;
		float distance = 0.0f;
		int nearest = -1;

		// Find the closest centroid
		for (int i = 0; i < centroids.length; i++) {
			distance = point.distance(centroids[i], p);
			if (distance < minDist) {
				nearest = i;
				minDist = distance;
			}
		}

		centroid.set(nearest);
		context.write(centroid, point);
	}
}
```

**RESULT:-**

Thus, the Program has been successfully completed

**EX.NO:-6** **DATE:-14/05/2024**

## Installation of Hive along with practice examples.

**Aim:-** To Installation of Hive along with practice examples.

## ALGORITHM:

### Step I:

Download java (JDK <latest version> - X64.tar.gz) by visiting the following link JavaSE Upgrade (oracle.com)

Then jdk-7u71-linux-x64.tar.gz will be downloaded onto your system.

### Step II:

Generally you will find the downloaded java file in the Downloads folder. Verify it and extract the jdk-7u71-linux-x64.gz file using the following commands.

```
$ cd Downloads/
$ ls
jdk-7u71-linux-x64.gz
$ tar zxf jdk-7u71-linux-x64.gz
$ ls
jdk1.7.0_71 jdk-7u71-linux-x64.gz
```

### Step III:

To make java available to all the users, you have to move it to the location "/usr/local/". Open root, and type the following commands.

```
$ su
password:
# mv jdk1.7.0_71 /usr/local/
# exit
```

### Step IV:

For setting up PATH and JAVA_HOME variables, add the following commands to ~/.bashrc file.

```
export JAVA_HOME=/usr/local/jdk1.7.0_71
export PATH=$PATH:$JAVA_HOME/bin
```

Row apply all the changes into the current running system.

```
$ source ~/.bashrc
```

## Step V:

Use the following commands to configure java alternatives:

```
# alternatives --install /usr/bin/java/java/usr/local/java/bin/java 2

# alternatives --install /usr/bin/javac/javac/usr/local/java/bin/javac 2

# alternatives --install /usr/bin/jar/jar/usr/local/java/bin/jar 2

# alternatives --set java/usr/local/java/bin/java

# alternatives --set javac/usr/local/java/bin/javac

# alternatives --set jar/usr/local/java/bin/jar
```

Now verify the installation using the command java -version from the terminal as explained above.

**RESULT :-**

Thus, the Program has been successfully completed