```java
/*
// Name:- Gyana Ranjan Tripathy
// UTA ID:- 1002079675
// CSE-6332-004-Cloud Computing and Bigdata
// Project - 1 : Matrix Multiplication on MapReduce
*/
import java.io.DataInput;
import java.io.DataOutput;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Scanner;

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.*;
import org.apache.hadoop.mapreduce.lib.output.*;
import org.apache.hadoop.util.ReflectionUtils;

public class Multiply {
    //this class helps implementation of the 'writable' interface that allows serialization
& deserialization of its data
    public static class Elemnt implements Writable {
        short tag;
        int index;
        double value;

        public Elemnt() {
        }

        public Elemnt(short tag, int index, double value) {
            this.tag = tag;
            this.index = index;
            this.value = value;
        }

        @Override
        public void write(DataOutput out) throws IOException {
            out.writeShort(tag);
            out.writeInt(index);
            out.writeDouble(value);
        }

        @Override
        public void readFields(DataInput in) throws IOException {
            tag = in.readShort();
            index = in.readInt();
            value = in.readDouble();
        }

        @Override
        public String toString() {
            return tag + "\t" + index + "\t" + value;
        }
    }

    // This method inPairs implements the writablecomparable interface to allow for sorting
    public static class inPairs implements WritableComparable<inPairs> {
        int i;
        int j;

        public inPairs() {
```

```
        }

        public inPairs(int i, int j) {
            this.i = i;
            this.j = j;
        }

        @Override
        public void write(DataOutput out) throws IOException {
            out.writeInt(i);
            out.writeInt(j);
        }

        @Override
        public void readFields(DataInput in) throws IOException {
            i = in.readInt();
            j = in.readInt();
        }

        @Override
        public int compareTo(inPairs inPairs) {
            if (i != inPairs.i) {
                return Integer.compare(i, inPairs.i);
            } else {
                return Integer.compare(j, inPairs.j);
            }
        }

        @Override
        public String toString() {
            return i + "\t" + j;
        }
    }
    // First Map task
    public static class FirstMapperTaskM extends Mapper<Object, Text, IntWritable, Elemnt> {
        @Override
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            Scanner input = new Scanner(value.toString()).useDelimiter("\\s+");
            int i = input.nextInt();
            int j = input.nextInt();
            double v = input.nextDouble();
            context.write(new IntWritable(j), new Elemnt((short) 0, i, v));
            input.close();
        }
    }

    // Second Map task
    public static class FirstMapperTaskN extends Mapper<Object, Text, IntWritable, Elemnt> {
        @Override
        public void map(Object key, Text value, Context context) throws IOException,
InterruptedException {
            Scanner input = new Scanner(value.toString()).useDelimiter("\\s+");
            int i = input.nextInt();
            int j = input.nextInt();
            double v = input.nextDouble();
            context.write(new IntWritable(i), new Elemnt((short) 1, j, v));
            input.close();
        }
    }

    // First Reducer task
    public static class FirstReducerTask extends Reducer<IntWritable, Elemnt, inPairs,
DoubleWritable> {
```

```
        @Override
        public void reduce(IntWritable key, Iterable<Elemnt> values, Context context)
                throws IOException, InterruptedException {
            ArrayList<Elemnt> m = new ArrayList<>();
            ArrayList<Elemnt> n = new ArrayList<>();
            Configuration config = context.getConfiguration();
            for (Elemnt e : values) {
                Elemnt tmp = ReflectionUtils.newInstance(Elemnt.class, config);
                ReflectionUtils.copy(config, e, tmp);
                if (e.tag == 0) {
                    m.add(tmp);
                } else if (e.tag == 1) {
                    n.add(tmp);
                }
            }
            for (Elemnt e1 : m) {
                for (Elemnt e2 : n) {
                    inPairs p = new inPairs(e1.index, e2.index);
                    double multiplyResult = e1.value * e2.value;
                    context.write(p, new DoubleWritable(multiplyResult));
                }
            }
        }
    }

    // Second Map task
    public static class SecondMapperTask extends Mapper<inPairs, DoubleWritable, inPairs,
DoubleWritable> {
        @Override
        public void map(inPairs key, DoubleWritable value, Context context) throws
IOException, InterruptedException {
            context.write(key, value);
        }
    }

    // Second Reducer task
    public static class SecondReducerTask extends Reducer<inPairs, DoubleWritable, inPairs,
DoubleWritable> {
        @Override
        public void reduce(inPairs key, Iterable<DoubleWritable> values, Context context)
                throws IOException, InterruptedException {
            double sum = 0.0;
            for (DoubleWritable value : values) {
                sum += value.get();
            }
            context.write(key, new DoubleWritable(sum));
        }
    }

    // Delete Record Function
    public static boolean deleteRecord(File dir) {
        if (dir.exists()) {
            File[] files = dir.listFiles();
            if (files != null) {
                for (File file : files) {
                    if (file.isDirectory()) {
                        deleteRecord(file);
                    } else {
                        file.delete();
                    }
                }
            }
        }
        return dir.delete();
```

```
    }

    // Main Function

    public static void main(String[] args) throws Exception {
        File tmp = new File("Intermediate");
        deleteRecord(tmp);
        Configuration config = new Configuration();
        Job firstJob = Job.getInstance(config, "firstJob");
        firstJob.setJarByClass(Multiply.class);
        firstJob.setMapOutputKeyClass(IntWritable.class);
        firstJob.setMapOutputValueClass(Elemnt.class);
        MultipleInputs.addInputPath(firstJob, new Path(args[0]), TextInputFormat.class,
FirstMapperTaskM.class);
        MultipleInputs.addInputPath(firstJob, new Path(args[1]), TextInputFormat.class,
FirstMapperTaskN.class);
        firstJob.setReducerClass(FirstReducerTask.class);
        firstJob.setOutputKeyClass(inPairs.class);
        firstJob.setOutputValueClass(DoubleWritable.class);
        firstJob.setOutputFormatClass(SequenceFileOutputFormat.class);
        SequenceFileOutputFormat.setOutputPath(firstJob, new Path(args[2]));
        firstJob.waitForCompletion(true);

        Job secondJob = Job.getInstance(config, "secondJob");
        secondJob.setJarByClass(Multiply.class);
        secondJob.setMapperClass(SecondMapperTask.class);
        secondJob.setMapOutputKeyClass(inPairs.class);
        secondJob.setMapOutputValueClass(DoubleWritable.class);
        secondJob.setInputFormatClass(SequenceFileInputFormat.class);
        SequenceFileInputFormat.addInputPath(secondJob, new Path(args[2]));
        secondJob.setReducerClass(SecondReducerTask.class);
        secondJob.setOutputKeyClass(inPairs.class);
        secondJob.setOutputValueClass(DoubleWritable.class);
        secondJob.setOutputFormatClass(TextOutputFormat.class);
        FileOutputFormat.setOutputPath(secondJob, new Path(args[3]));
        secondJob.waitForCompletion(true);
    }
}
```