```
import java.io.*;
import java.util.*;
import javax.security.auth.login.Configuration;
import javax.tools.Tool;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.util.*;
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.*;

class Pair implements WritableComparable<Pair> {
    public int i;
    public int j;

    Pair() {
    }

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

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

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

    }

    @Override
    public int compareTo(Pair other) {
        if (i > other.i) {
            return 1;
        } else if (i < other.i) {
            return -1;
        } else {
            if (j > other.j) {
                return 1;
            } else if (j < other.j) {
                return -1;
            }
        }
        return 0;
    }

    public String toString() {
        return i + "," + j + ",";
    }

}

class Element implements Writable {
    public short tag;
```

```
    public First fr;
    public Second sc;

    Element() {
    }

    Element(First j) {
        tag = 0;
        fr = j;
    }

    Element(Second k) {
        tag = 1;
        sc = k;
    }

    public void write(DataOutput out) throws IOException {
        out.writeShort(tag);
        if (tag == 0)
            fr.write(out);
        else
            sc.write(out);
    }

    public void readFields(DataInput in) throws IOException {
        tag = in.readShort();
        if (tag == 0) {
            fr = new First();
            fr.readFields(in);
        } else {
            sc = new Second();
            sc.readFields(in);
        }

    }

}

class First implements Writable {
    public int row;
    public int column;
    public double erb;

    First() {
    }

    First(int a, int b, double c) {
        row = a;
        column = b;
        erb = c;
    }

    public void write(DataOutput out) throws IOException {
        out.writeInt(row);
        out.writeInt(column);
        out.writeDouble(erb);
    }

    public void readFields(DataInput in) throws IOException {
```

```
        row = in.readInt();
        column = in.readInt();
        erb = in.readDouble();
    }

}

class Second implements Writable {
    public int row;
    public int column;
    public double erb;

    Second() {
    }

    Second(int a, int b, double c) {
        row = a;
        column = b;
        erb = c;
    }

    public void write(DataOutput out) throws IOException {
        out.writeInt(row);
        out.writeInt(column);
        out.writeDouble(erb);
    }

    public void readFields(DataInput in) throws IOException {
        row = in.readInt();
        column = in.readInt();
        erb = in.readDouble();
    }

}

public class Multiply {

    public static class MMatrixMapper extends Mapper<Object, Text, IntWritable,
Element> {
        @Override
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            Scanner s = new Scanner(value.toString()).useDelimiter(",");
            First f = new First(s.nextInt(), s.nextInt(), s.nextDouble());
            context.write(new IntWritable(f.column), new Element(f));
            s.close();
        }
    }

    public static class NMatrixMapper extends Mapper<Object, Text, IntWritable,
Element> {
        @Override
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            Scanner s = new Scanner(value.toString()).useDelimiter(",");
            Second g = new Second(s.nextInt(), s.nextInt(), s.nextDouble());
            context.write(new IntWritable(g.row), new Element(g));
            s.close();
        }
```

```
    }

    public static class MultReducer extends Reducer<IntWritable, Element, Pair,
DoubleWritable> {
        static Vector<First> frs = new Vector<First>();
        static Vector<Second> srs = new Vector<Second>();

        @Override
        public void reduce(IntWritable key, Iterable<Element> values, Context
context)
                throws IOException, InterruptedException {
            frs.clear();
            srs.clear();
            for (Element v : values)
                if (v.tag == 0)
                    frs.add(v.fr);
                else
                    srs.add(v.sc);
            double sv;
            for (First gh : frs)
                for (Second sg : srs) {
                    sv = gh.erb * sg.erb;
                    context.write(new Pair(gh.row, sg.column), new
DoubleWritable(sv));
                }
        }
    }

    public static class SecondMapper extends Mapper<Object, Text, Pair,
DoubleWritable> {
        @Override
        public void map(Object key, Text value, Context context) throws
IOException, InterruptedException {
            Scanner s = new Scanner(value.toString()).useDelimiter(",");
            Pair pair = new Pair(s.nextInt(), s.nextInt());
            double sd = Double.parseDouble(s.next().trim());
            context.write(pair, new DoubleWritable(sd));
        }
    }

    public static class LastReducer extends Reducer<Pair, DoubleWritable, Text,
NullWritable> {
        @Override
        public void reduce(Pair pair, Iterable<DoubleWritable> values, Context
context)
                throws IOException, InterruptedException {
            double m = 0.0;
            for (DoubleWritable v : values)
                m = m + v.get();
            String lk = pair.i + "," + pair.j + "," + m;
            context.write(new Text(lk), NullWritable.get());

        }
    }

    public static void main(String[] args) throws Exception {
        Job job = Job.getInstance();
        job.setJobName("first map reduce job");
        job.setJarByClass(Multiply.class);
```

```
        job.setOutputKeyClass(Pair.class);
        job.setOutputValueClass(DoubleWritable.class);
        job.setMapOutputKeyClass(IntWritable.class);
        job.setMapOutputValueClass(Element.class);
        job.setReducerClass(MultReducer.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class,
MMatrixMapper.class);
        MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class,
NMatrixMapper.class);
        FileOutputFormat.setOutputPath(job, new Path(args[2]));
        job.waitForCompletion(true);

        Job job2 = Job.getInstance();
        job2.setJobName("second map reduce job");
        job2.setJarByClass(Multiply.class);
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(NullWritable.class);
        job2.setMapOutputKeyClass(Pair.class);
        job2.setMapOutputValueClass(DoubleWritable.class);
        job2.setMapperClass(SecondMapper.class);
        job2.setReducerClass(LastReducer.class);
        job2.setInputFormatClass(TextInputFormat.class);
        job2.setOutputFormatClass(TextOutputFormat.class);
        FileInputFormat.setInputPaths(job2, new Path(args[2]));
        FileOutputFormat.setOutputPath(job2, new Path(args[3]));
        job2.waitForCompletion(true);
    }
}
```