```java
import java.io.*;
import java.util.*;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.util.*;
import org.apache.hadoop.mapreduce.lib.input.*;
import org.apache.hadoop.mapreduce.lib.output.*;

//Matrix A
class MatrixA implements Writable{
    public int Ai,Aj;
    public float Val_A;

    MatrixA() {}

    MatrixA(int a, int b, float c) {
        Ai = a;
        Aj = b;
        Val_A = c;

    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {

         dataOutput.writeInt(Ai);
         dataOutput.writeInt(Aj);
         dataOutput.writeFloat(Val_A);

    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        Ai = dataInput.readInt();
        Aj = dataInput.readInt();
        Val_A = dataInput.readFloat();
    }
}
//Matrix B

class MatrixB implements Writable{
    public int Bi,Bj;
    public float Val_B;

    MatrixB() {}

    MatrixB(int a, int b, float c) {
        Bi = a;
        Bj = b;
        Val_B = c;

    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {

        dataOutput.writeInt(Bi);
        dataOutput.writeInt(Bj);
        dataOutput.writeFloat(Val_B);

    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        Bi = dataInput.readInt();
        Bj = dataInput.readInt();
        Val_B = dataInput.readFloat();
    }
}

class MatrixAB implements Writable{
    public int ABi,ABj;
    public float Val_AB;


    MatrixAB() {}
    MatrixAB (int a, int b, float c) {
        ABi = a;
        ABj = b;
        Val_AB = c;
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeInt(ABi);
        dataOutput.writeInt(ABj);
        dataOutput.writeFloat(Val_AB);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        ABi = dataInput.readInt();
        ABj = dataInput.readInt();
        Val_AB = dataInput.readFloat();

    }
}
//Class Elem

class Elem implements Writable{

    public char tag;
    public MatrixA a1;
    public MatrixB b1;

    Elem () {}
    Elem (MatrixA a2){
        tag = 'A';
        a1 = a2;
    }
    Elem (MatrixB b2){
        tag = 'B';
        b1 = b2;
    }
    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeChar(tag);
        if(tag == 'A'){
            a1.write(dataOutput);
        }
        else {
            b1.write(dataOutput);
        }

    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        tag = dataInput.readChar();
        if (tag == 'A'){
            a1 = new MatrixA();
            a1.readFields(dataInput);
        }
        else {
            b1 = new MatrixB();
            b1.readFields(dataInput);
        }
    }

}

class Matrix_Result implements Writable{

    public MatrixAB ab1;
    public int abi, abj;
    public float abVal;

    Matrix_Result() {}
    Matrix_Result(MatrixAB ab2) {
        ab1 = ab2;
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeInt(abi);
        dataOutput.writeInt(abj);
        dataOutput.writeFloat(abVal);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        abi = dataInput.readInt();
        abj = dataInput.readInt();
        abVal = dataInput.readFloat();
    }
}

public class Multiply extends Configured implements Tool {

    //FIRST MAPREDUCE

//First Mapper
```

```
    public static class Mapper_First extends Mapper<LongWritable,Text,Text,Elem>{
        @Override
        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException{
            //Get next line
            String next_line = value.toString();
            String[] val = next_line.split(",");
            MatrixA a1 = new
MatrixA(Integer.parseInt(val[0]),Integer.parseInt(val[1]),Float.parseFloat(val[2]))
;
            Text key_pair = new Text();
            key_pair.set(String.valueOf(a1.Aj));
            context.write(key_pair, new Elem(a1));

        }

    }
    //Second Mapper

    public static class Mapper_Second extends Mapper<LongWritable,Text,Text,Elem>{
        @Override
        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException{
            String next_line = value.toString();
            String[] val = next_line.split(",");
            MatrixB b1 = new
MatrixB(Integer.parseInt(val[0]),Integer.parseInt(val[1]),Float.parseFloat(val[2]))
;
            Text key_pair = new Text();
            key_pair.set(String.valueOf(b1.Bi));
            context.write(key_pair, new Elem(b1));
        }

    }



    public static class Reducer_multiply extends Reducer<Text,Elem,Text,Text>{
        @Override
        public void reduce(Text key, Iterable<Elem> value, Context context)throws
IOException, InterruptedException{
            String input_reducer = value.toString();
            String[] input_values;

            //getting inputs to reducer and dividing into two parts
            Vector<MatrixA> A_matrix = new Vector<MatrixA>();
            Vector<MatrixB> B_matrix = new Vector<MatrixB>();
            Text key_pair = new Text();
            Text value_pair = new Text();
            double value_elem;

            A_matrix.clear();
            B_matrix.clear();

            for (Elem v : value)
            {
                if(v.tag == 'A')
                {
                    A_matrix.add(v.a1);
                }else {
                    B_matrix.add(v.b1);
                }
            }

            //Multiply values
            for(MatrixA v: A_matrix)
            {
                for(MatrixB v1:B_matrix)
                {
                    key_pair.set(key + ",");
                    value_elem = v.Val_A * v1.Val_B;
                    value_pair.set(v.Ai + "," + v1.Bj + "," + value_elem);
                    context.write(key_pair, value_pair);
                }
            }
        }
    }

    //SECOND MAPREDUCE
//Third Mapper

    public static class Mapper_Third extends Mapper<LongWritable,Text,Text,Text>{
        @Override
        public void map(LongWritable key, Text value, Context context) throws
IOException, InterruptedException{
            String next_line = value.toString();
            String[] val = next_line.split(",");
            Text key_pair = new Text();
            key_pair.set(val[1] + "," + val[2]);
            Text value_pair = new Text();
            value_pair.set(val[3]);
            context.write(key_pair, value_pair);
        }
    }


    public static class Reducer_summation extends Reducer<Text,Text,Text,Text>{
        @Override
        public void reduce(Text key, Iterable<Text> value, Context context)throws
IOException, InterruptedException{
            float output = 0;

            //Set Outputs
            Text key_pair = new Text();
            Text value_pair = new Text();

            for(Text v: value)
            {
                output += Float.parseFloat(v.toString());
            }
            key_pair.set(key + ",");
            value_pair.set(String.valueOf(output));
            context.write(key_pair, value_pair);

        }
    }

    public static void main (String[] args) throws Exception {
        int big = ToolRunner.run(new Configuration(), new Multiply(), args);
        System.exit(big);
    }

    public int run(String [] args) throws Exception {


        Configuration j1 = new Configuration();

        Job FirstJob = Job.getInstance(j1,"MatMut");
        FirstJob.setJarByClass(Multiply.class);
        FirstJob.setOutputKeyClass(Text.class);
        FirstJob.setOutputValueClass(Elem.class);

        MultipleInputs.addInputPath(FirstJob,new
Path(args[0]),TextInputFormat.class,Mapper_First.class);
        MultipleInputs.addInputPath(FirstJob,new
Path(args[1]),TextInputFormat.class,Mapper_Second.class);

        FileOutputFormat.setOutputPath(FirstJob,new Path(args[2], "result"));
        FirstJob.setReducerClass(Reducer_multiply.class);

        FirstJob.waitForCompletion(true);



        Configuration j2 = new Configuration();

        Job SecondJob = Job.getInstance(j2,"Result matrix");
        SecondJob.setJarByClass(Multiply.class);
        SecondJob.setOutputKeyClass(Text.class);
        SecondJob.setOutputValueClass(Text.class);

        SecondJob.setMapperClass(Mapper_Third.class);
        SecondJob.setReducerClass(Reducer_summation.class);

        FileInputFormat.addInputPath(SecondJob,new Path(args[2],"result"));
        FileOutputFormat.setOutputPath(SecondJob,new Path(args[3]));

        return SecondJob.waitForCompletion(true) ? 0 : 1;



    }
}
```