```
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 Bj,Bk;
    public float Val_B;

    MatrixB() {}

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

    }

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

```
        dataOutput.writeInt(Bj);
        dataOutput.writeInt(Bk);
        dataOutput.writeFloat(Val_B);

    }

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

class Pair implements Writable{
    public int i,j;
//    public float Val_AB;


    Pair() {}
    Pair (int a, int b) {
        i = a;
        j = b;
//        Val_AB = c;
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeInt(i);
        dataOutput.writeInt(j);
//        dataOutput.writeFloat(Val_AB);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        i = dataInput.readInt();
        j = dataInput.readInt();
//        Val_AB = dataInput.readFloat();

    }
}
//Class Elem

class Elem implements Writable{

    public int tag;
    public MatrixA a1;
    public MatrixB b1;

    Elem () {}
    Elem (MatrixA a2){
        tag = 0;
        a1 = a2;
    }
    Elem (MatrixB b2){
        tag = 1;
        b1 = b2;
    }
```

```
    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeChar(tag);
        if(tag == 0){
            a1.write(dataOutput);
        }
        else {
            b1.write(dataOutput);
        }

    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        tag = dataInput.readChar();
        if (tag == 0){
            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,Pair,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(0,Integer.parseInt(val[1]),Float.parseFloat(val[2]));
            int p = 1000;
            Text key_pair = new Text();
            for (int k = 0; k < p; k++) {

//                key_pair.set(String.valueOf(a1.Ai) + "," + k);
                context.write(new Pair(Integer.parseInt(val[0]), k), new Elem(a1));
            }
        }

    }
    //Second Mapper

    public static class Mapper_Second extends Mapper<LongWritable,Text,Pair,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]));
            MatrixB b1 = new MatrixB(1, Integer.parseInt(val[0]),
Float.parseFloat(val[2]));
            Text key_pair = new Text();
            int m = 1000;

            for (int i = 0; i < m; i++) {

//                key_pair.set(String.valueOf(b1.Bi) + "," + k);
//                key_pair.set( i + "," + String.valueOf(b1.Bj));
//                outputValue.set("0" + "," + val[0] + "," + val[2]);
                context.write(new Pair(i,Integer.parseInt(val[1])), new Elem(b1));
            }

        }
    }



    public static class ReducerM extends Reducer<Pair,Elem,Text,Text>{
        @Override
        public void reduce(Pair 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>();
//
//
//            A_matrix.clear();
//            B_matrix.clear();
//
//            for (Elem v : value)
//            {
//                if(v.tag == 0)
//                {
//                    A_matrix.add(v.a1);
//                }else
//                {
//                    B_matrix.add(v.b1);
//                }
//            }

            //key=(i,k),
            //Values = [(M/N,j,V/W),..]
            //Making  Hashmap

            HashMap<Integer,Float> hashA = new HashMap<Integer,Float>();
            HashMap<Integer,Float> hashB = new HashMap<Integer,Float>();



            for (Elem val : value) {
//                value1 = val.toString().split(",");
                if (val.tag == 0) {
                    hashA.put(val.a1.Aj, val.a1.Val_A);
                } else {
                    hashB.put(val.b1.Bj, val.b1.Val_B);
                }
            }
            int n = Integer.parseInt(context.getConfiguration().get("n"));
            float result = 0.0f;
            float m_ij;
            float n_jk;
            for (int j = 0; j < n; j++) {
                m_ij = hashA.containsKey(j) ? hashA.get(j) : 0.0f;
                n_jk = hashB.containsKey(j) ? hashB.get(j) : 0.0f;
                result += m_ij * n_jk;
            }
            if (result != 0.0f) {
                context.write(null,
                        new Text(key.toString() + "," + Float.toString(result)));
            }
        }
    }

            // Calculate sum of products for each key (i, k)

//            Map<Pair, Elem> result = new HashMap<>();
//
//            for (String key : hashA.keySet()) {
//                List<Elem> valuesM = hashA.get(key);
//                List<Elem> valuesN = hashB.get(key);
//
//                // Sort values of M and N by j
```

```
//                Map<Integer, Integer> mapM = new HashMap<>();
//                Map<Integer, Integer> mapN = new HashMap<>();
//                for (int j = 0; j < valuesM.size(); j++) {
//                    mapM.put(valuesM.get(j), j);
//                    mapN.put(valuesN.get(j), j);
//                }
//
//                TreeMap<Integer, Integer> sortedMapM = new TreeMap<>(mapM);
//                TreeMap<Integer, Integer> sortedMapN = new TreeMap<>(mapN);
//                List<Integer> sortedValuesM = new ArrayList<>();
//                List<Integer> sortedValuesN = new ArrayList<>();
//                for (int index : sortedMapM.values()) {
//                    sortedValuesM.add(valuesM.get(index));
//
sortedValuesN.add(valuesN.get(sortedMapM.get(valuesM.get(index))));
//                }
//
//                // Calculate sum of products for this key (i, k)
//                int sum = 0;
//                for (int j = 0; j < sortedValuesM.size(); j++) {
//                    sum += sortedValuesM.get(j) * sortedValuesN.get(j);
//                }
//                result.put(new Pair<>(key.substring(0, 2), key.substring(2)),
sum);
//            }
//
//            // Print result
//            for (Pair<String, String> key : result.keySet()) {
//                System.out.println("(" + key.getFirst() + ", " + key.getSecond()
+ ") => " + result.get(key));
//            }
//        }
//    }



    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();
        // M is an m-by-n matrix; N is an n-by-p matrix.
//        conf.set("m", "1000");
//        conf.set("n", "100");
//        conf.set("p", "1000");

        Job FirstJob = Job.getInstance(j1,"MatMut");
        FirstJob.setJarByClass(Multiply.class);
        FirstJob.setOutputKeyClass(Pair.class);
        FirstJob.setOutputValueClass(Elem.class);
        FirstJob.setMapOutputValueClass(IntWritable.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(ReducerM.class);

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



    }
}
```