```java
/**
 * Matrix Multiplication.
 * Consider matrices A (L,M)   (L lines, M cols)
 *              and B (M,N)   (M lines, N cols)
 * we want to compute resulting matrix
 *                    A x B = C (L,N)
 *
 * The input should be in a text file formatted as:
 * A : [0 1 2 3 4] [5 6 7 8 9]
 * B : [0 1 2] [3 4 5] [6 7 8] [9 10 11] [12 13 14]
 *
 * Stéphane Genaud, Nov 2013
 **/


import java.io.IOException;
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.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class Mm {


      public static class Map extends Mapper<LongWritable, Text, Text, Text> {

            /**
             * map : the map function implementation
             *
             * @param key
             * @param value  the data from input files, one line per map
                 invocation
             * @param contex a place to write the (key,value) pairs we want to
                 emit
             **/
            public void map(LongWritable key, Text value, Context context)
                throws IOException, InterruptedException {

                Text outputKey = new Text();
                Text outputValue = new Text();
                Configuration conf = context.getConfiguration();
                int L = Integer.parseInt(conf.get("L"));
                int M = Integer.parseInt(conf.get("M"));
                int N = Integer.parseInt(conf.get("N"));
                String [][] A = new String [L][M];
                String [][] B = new String [M][N];
                // separate matrix anem and values
                String [] matStruct = value.toString().split(":");
                matStruct[0]= matStruct[0].trim();
                matStruct[1]= matStruct[1].trim();
                //split values in tokens, each token being a matrix line
                String [] matLines = matStruct[1].split ("\\]");

                // A
                if (matStruct[0].equals("A")) {
                      // ------ first parse matrix values and store in an array
                      for (int i=0;i<L;i++) {
```

```java
                            // remove leading ']'
                            // (replace doesn't use a regexp so no protection)
                            matLines[i] = matLines[i].replace("[","").trim();
                            System.out.println(matLines[i]);
                            A[i] = matLines[i].split(" ");
                      }
                      // ------ second, emit the values needed for products
                      // emit (key,value)
                      for (int i=0;i<L;i++) {
                            for (int j=0;j<N;j++) {
                                for (int k=0;k<M;k++) {
                                      System.out.println("(C_"+i+"_"+j + ", A_"+i
                                          +","+k+")");
                                      outputKey.set("C_"+i+"_"+j);
                                      // values neeed to identify : matrix,
                                          position, value to multiply
                                      // i.e  (A,x,valA) will have to match (B,x,
                                          valB) to go valA*valB
                                      outputValue.set("A,"+ k +","+ A[i][k]);
                                      context.write(outputKey, outputValue);
                                }
                            }

                      }
                }
                // B
                if (matStruct[0].equals("B")) {
                      // fill in matrix
                      for (int i=0;i < ?? ;i++) {

                          ????
                          ????
                          ????


                      }
                      // emit (key,value)
                      for (int i=0;i< ??? ;i++) {
                            for (int j=0;j< ??? ;j++) {
                                for (int k=0;k< ??? ;k++) {

                                    ????
                                    ????
                                    ????

                                }
                            }
                      }
                }
          }
}

public static class Reduce extends Reducer<Text, Text, Text, Text> {

    /**
     * reduce
     * @param key the key as Text
     * @param values the list of values associated to key
     * @param context the place to store outputs (goes to results)
     **/

      public void reduce(Text key, Iterable<Text> values, Context context)
          throws IOException, InterruptedException {
            String [] value;
            int M = Integer.parseInt(context.getConfiguration().get("M"));
            Float [] valsA = new Float [M];
            Float [] valsB = new Float [M];
            System.out.print("k="+key.toString()+"->[");
            // e.g key: C_2_3 -> [("A,2,3.0");("A,2,4.0");("B,2,23.0") ...]
            for (Text val : values) {
                  value = val.toString().split(",");
                  int loc = Integer.parseInt(value[1]);
                  if (value[0].equals("A")) {
                        System.out.print("(A_"+value[1]+","+value[2].toString
                            ()+");");
                        valsA[loc] =  Float.parseFloat(value[2]);
                  } else {
                        System.out.print("(B_"+value[1]+","+value[2].toString
                            ()+");");
                        valsB[loc] =  Float.parseFloat(value[2]);
                  }
            }
            System.out.println("]");
            float result = 0.0f;
            float a_ij;
            float b_jk;
            for (int k = 0; k < ??? ; k++) {
                  result += valsA[k] * valsB[k];
            }
            context.write(null, new Text(key.toString() + "," + Float.
                toString(result)));
        }
    }

    public static void main(String[] args) throws Exception {

        if (args.length < 2) {
            System.err.println("Usage : hadoop jar Mm.jar Mm matrices.in res.
                out");
            System.exit(0);
        }


        Configuration conf = new Configuration();
        // A is an L-rows, M-cols matrix
        // B is an M-rows, N-cols matrix.
        conf.set("L", "2");
        conf.set("M", "5");
        conf.set("N", "3");

        Job job = new Job(conf, "MatrixMultiplication");
        job.setJarByClass(Mm.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(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.waitForCompletion(true);
    }
```

```
}
```