CSEE5590 Big Data Programming

In Class Programming –3 Report
(Jongkook Son)

Project Overview:

Overview of Hadoop and Map Reduce Paradigm. The Lesson focuses on map reduce coding exercises by actual implementation

Requirements/Task(s):

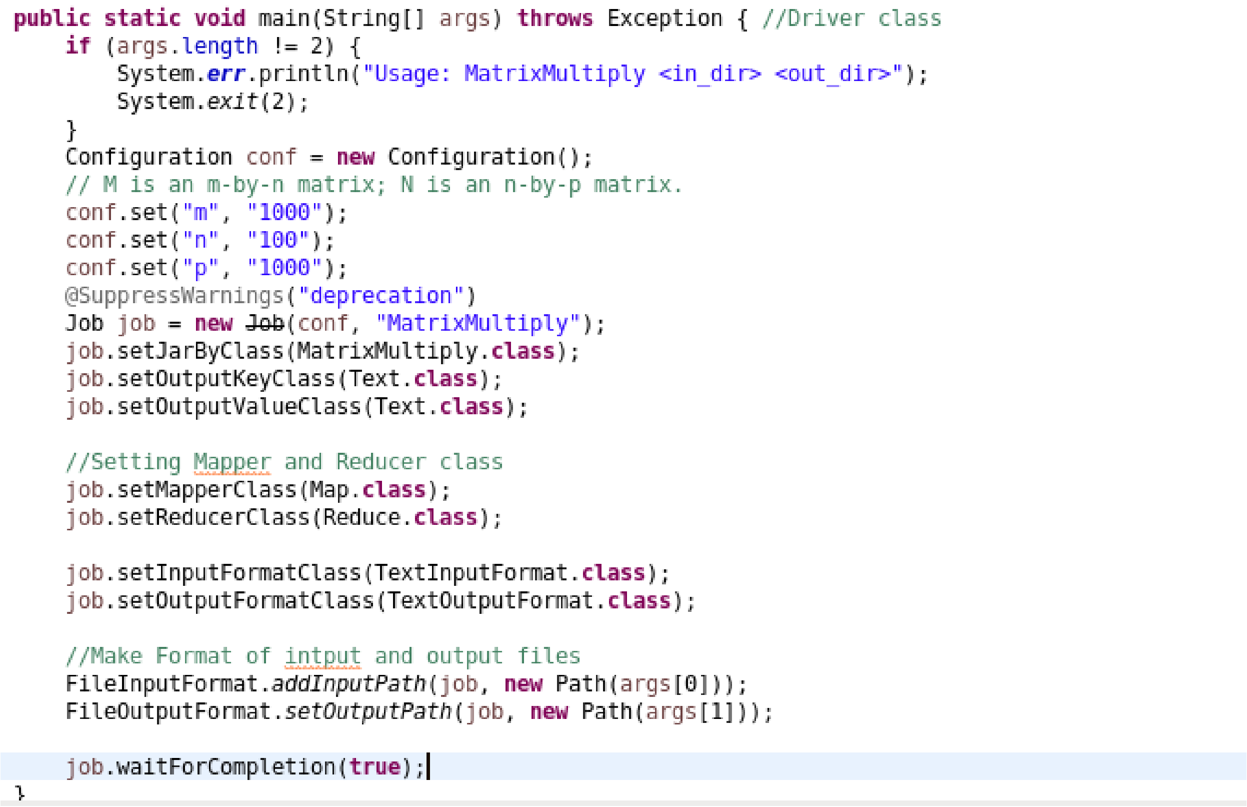
1. Matrix Multiplication in Map Reduce
Suppose we have a i x j matrix M, whose element in row i and column j will be denoted and a j x k matrix N whose element in row j and column k is donated by then the product P = MN will be i x k matrix P whose element in row i and column k will be donated by ,Create a Map-Reduce Program to perform the task of matrix multiplication Marks will be distributed between logic, implementation and UI (presentation in GITHUB Wiki)

What I learned in ICP:

I could have get the further insight process of the MapReduce. A MapReduce job usually splits the input data-set into independent block which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Thanks to this ICP3, I could have understand this process more deeply. In our code implementation,

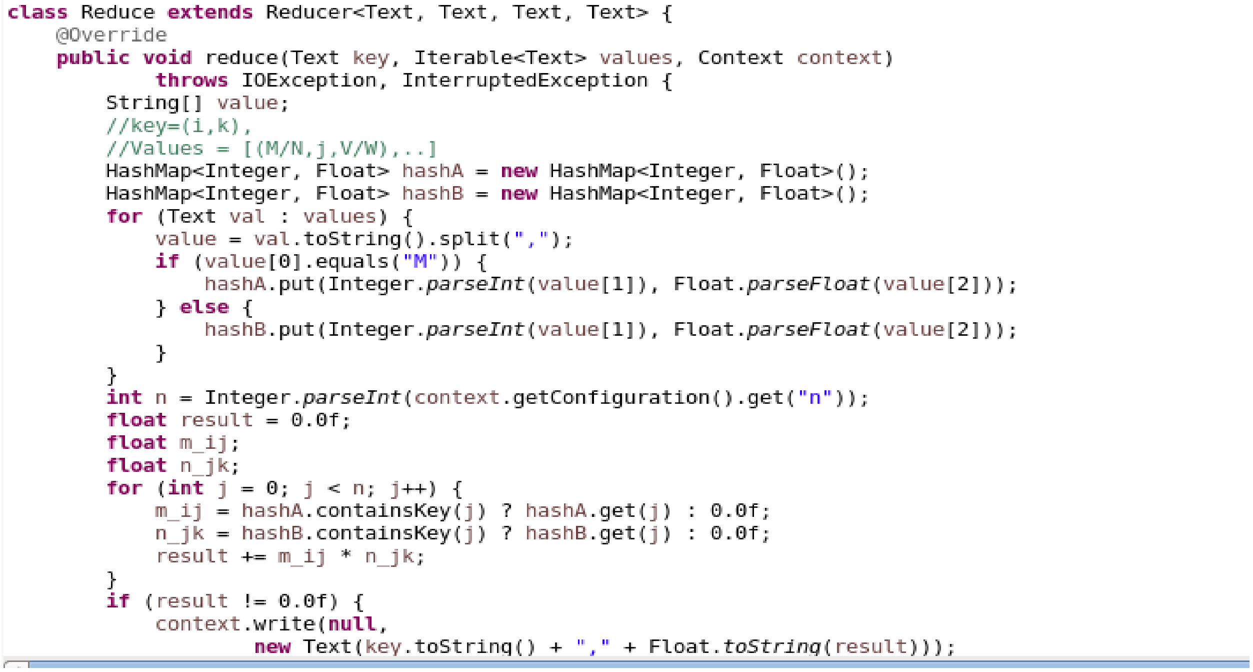
ICP description what was the task you were performing and Screen shots that shows the successful execution of each required step of your code

TASK



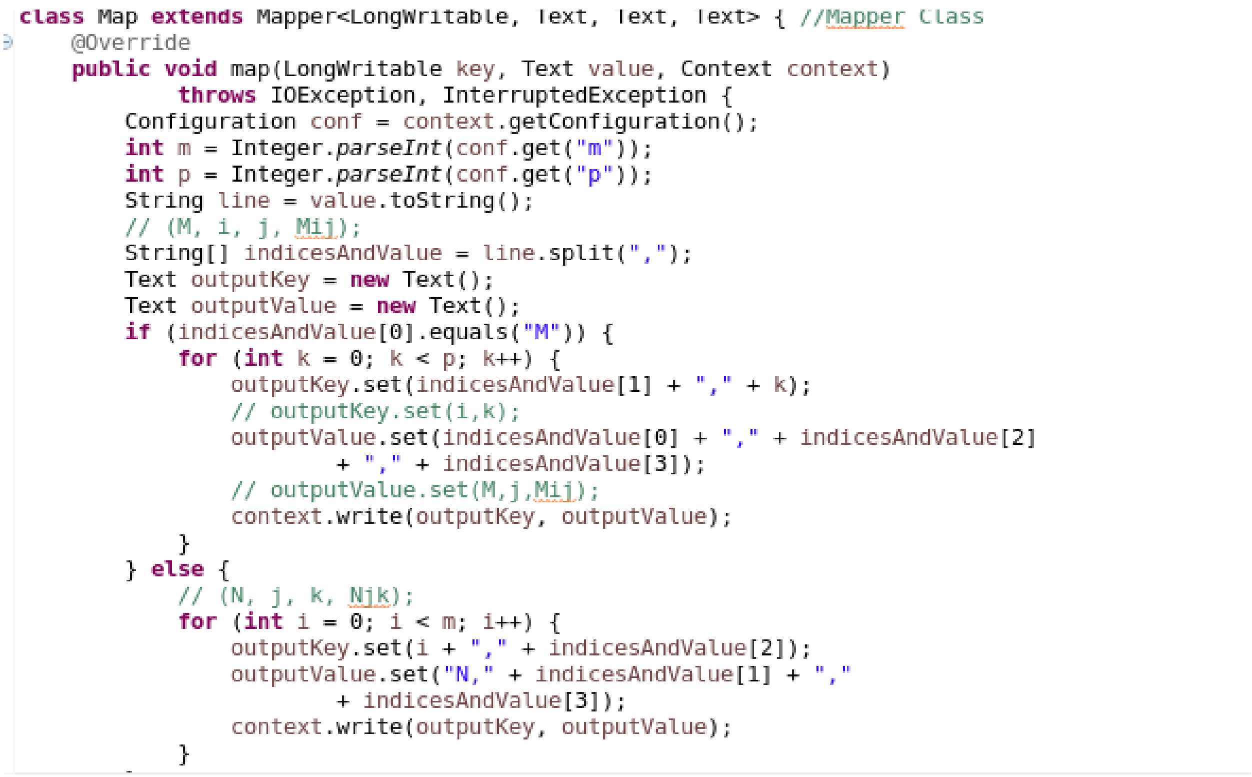
```
public static void main(String[] args) throws Exception { //Driver class
    if (args.length != 2) {
        System.err.println("Usage: MatrixMultiply <in_dir> <out_dir>");
        System.exit(2);
    }
    Configuration conf = new Configuration();
    // M is an m-by-n matrix; N is an n-by-p matrix.
    conf.set("m", "1000");
    conf.set("n", "100");
    conf.set("p", "1000");
    @SuppressWarnings("deprecation")
    Job job = new ~~Job~~(conf, "MatrixMultiply");
    job.setJarByClass(MatrixMultiply.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(Text.class);

    //Setting Mapper and Reducer class
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    //Make Format of intput and output files
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}
```

<Driver Class>



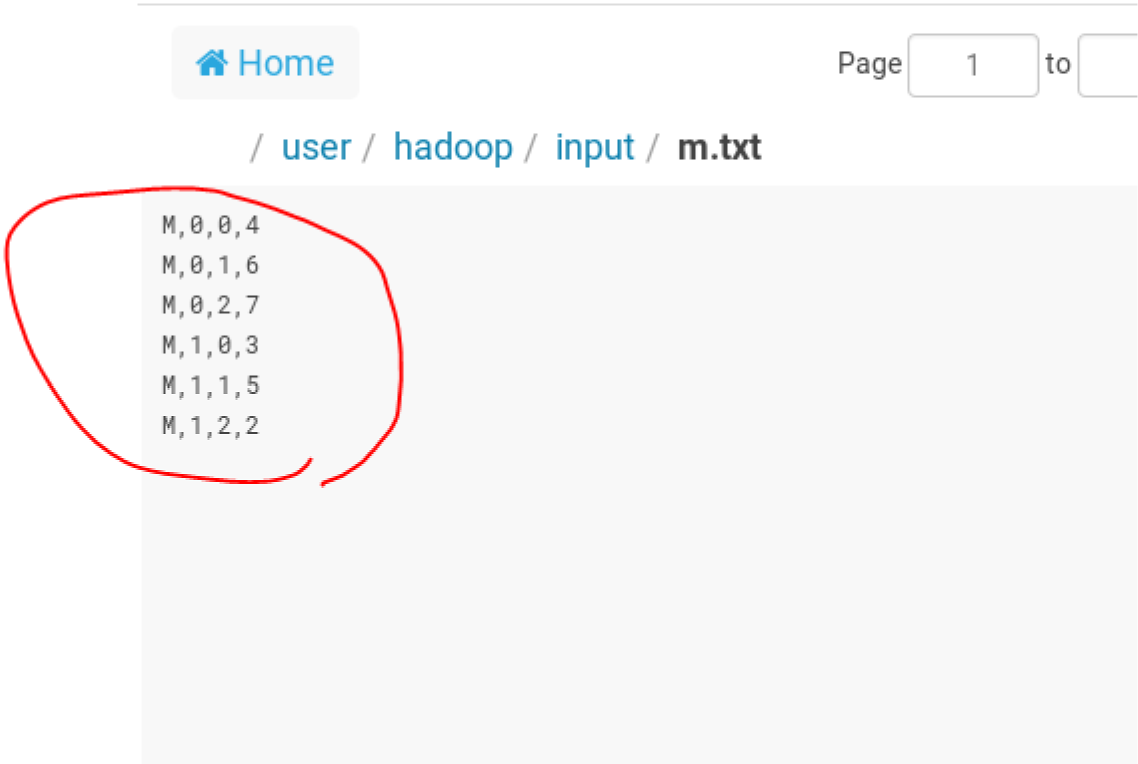
```
class Reduce extends Reducer<Text, Text, Text, Text> {
    @Override
    public void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {
        String[] value;
        //key=(i,k),
        //Values = [(M/N,j,V/W),..]
        HashMap<Integer, Float> hashA = new HashMap<Integer, Float>();
        HashMap<Integer, Float> hashB = new HashMap<Integer, Float>();
        for (Text val : values) {
            value = val.toString().split(",");
            if (value[0].equals("M")) {
                hashA.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            } else {
                hashB.put(Integer.parseInt(value[1]), Float.parseFloat(value[2]));
            }
        }
        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)));
```

<Mapper Class>



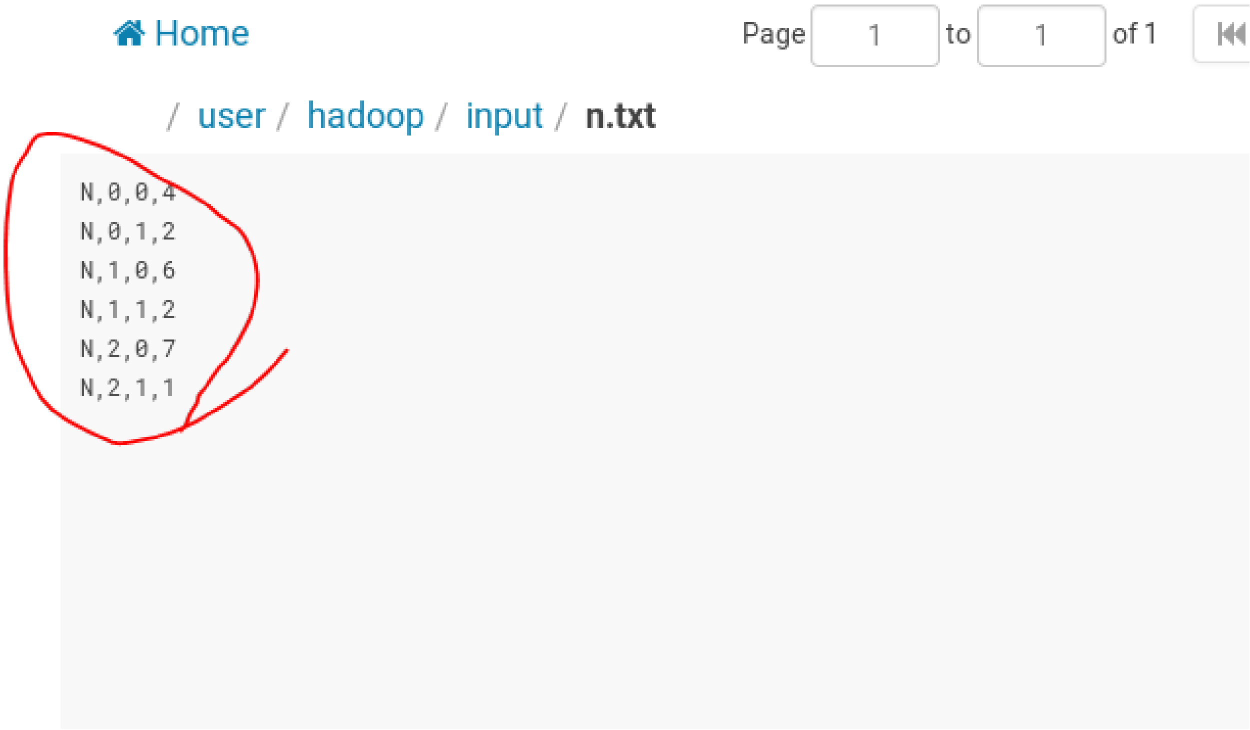
```
class Map extends Mapper<LongWritable, Text, Text, Text> { //Mapper Class
    @Override
    public void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {
        Configuration conf = context.getConfiguration();
        int m = Integer.parseInt(conf.get("m"));
        int p = Integer.parseInt(conf.get("p"));
        String line = value.toString();
        // (M, i, j, Mij);
        String[] indicesAndValue = line.split(",");
        Text outputKey = new Text();
        Text outputValue = new Text();
        if (indicesAndValue[0].equals("M")) {
            for (int k = 0; k < p; k++) {
                outputKey.set(indicesAndValue[1] + "," + k);
                // outputKey.set(i,k);
                outputValue.set(indicesAndValue[0] + "," + indicesAndValue[2]
                        + "," + indicesAndValue[3]);
                // outputValue.set(M,j,Mij);
                context.write(outputKey, outputValue);
            }
        } else {
            // (N, j, k, Njk);
            for (int i = 0; i < m; i++) {
                outputKey.set(i + "," + indicesAndValue[2]);
                outputValue.set("N," + indicesAndValue[1] + ","
                        + indicesAndValue[3]);
                context.write(outputKey, outputValue);
            }
```

<Reducer Class>



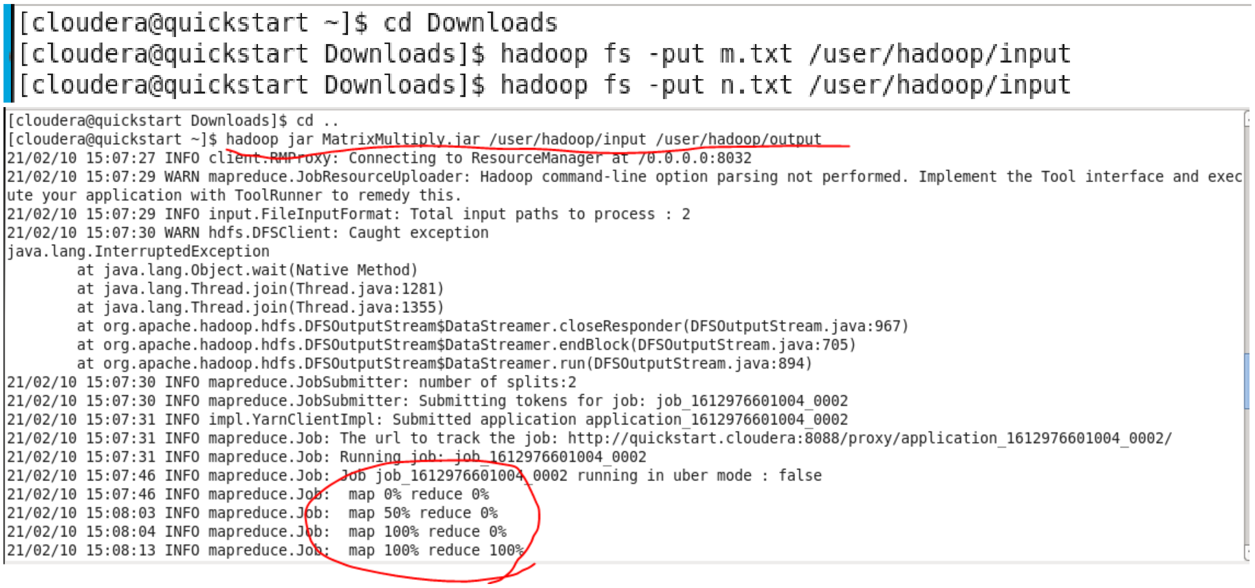
```
M,0,0,4
M,0,1,6
M,0,2,7
M,1,0,3
M,1,1,5
M,1,2,2
```

<INPUT M>



```
N,0,0,4
N,0,1,2
N,1,0,6
N,1,1,2
N,2,0,7
N,2,1,1
```

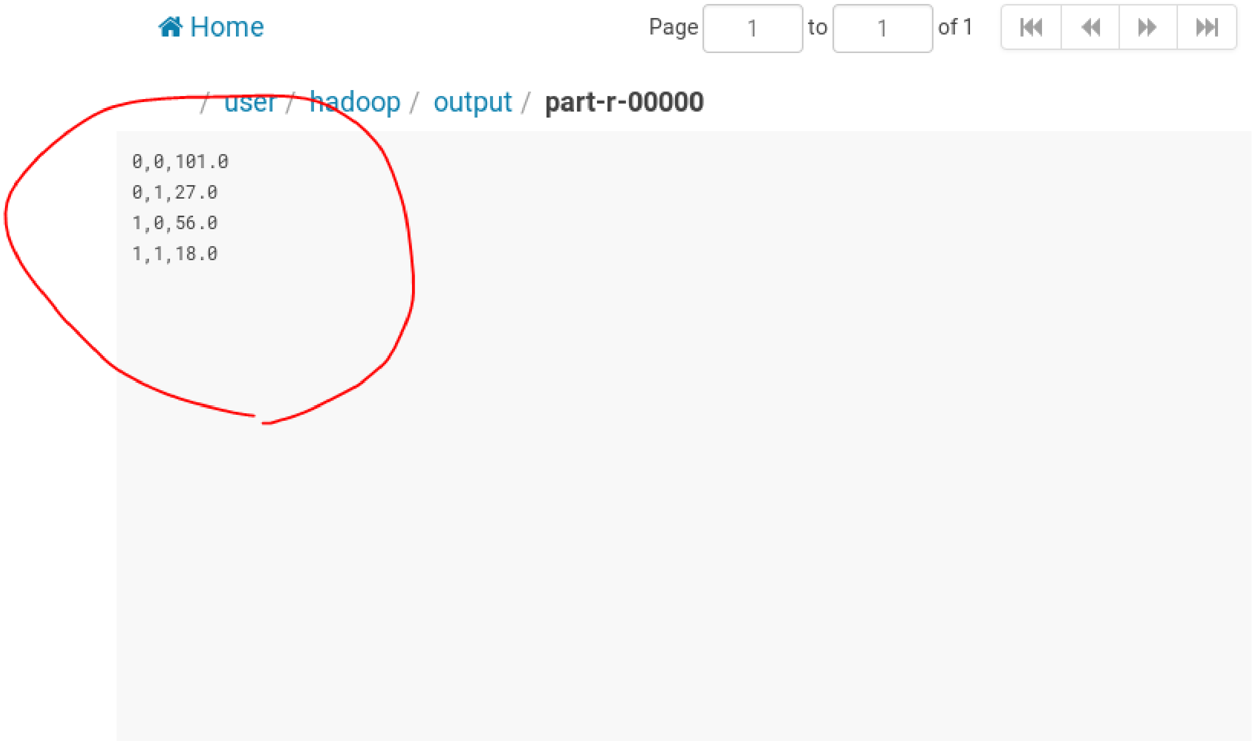
<INPUT N>



```
[cloudera@quickstart ~]$ cd Downloads
[cloudera@quickstart Downloads]$ hadoop fs -put m.txt /user/hadoop/input
[cloudera@quickstart Downloads]$ hadoop fs -put n.txt /user/hadoop/input
```

```
[cloudera@quickstart Downloads]$ cd ..
[cloudera@quickstart ~]$ hadoop jar MatrixMultiply.jar /user/hadoop/input /user/hadoop/output
21/02/10 15:07:27 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
21/02/10 15:07:29 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and exec
ute your application with ToolRunner to remedy this.
21/02/10 15:07:29 INFO input.FileInputFormat: Total input paths to process : 2
21/02/10 15:07:30 WARN hdfs.DFSClient: Caught exception
java.lang.InterruptedException
        at java.lang.Object.wait(Native Method)
        at java.lang.Thread.join(Thread.java:1281)
        at java.lang.Thread.join(Thread.java:1355)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.closeResponder(DFSOutputStream.java:967)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.endBlock(DFSOutputStream.java:705)
        at org.apache.hadoop.hdfs.DFSOutputStream$DataStreamer.run(DFSOutputStream.java:894)
21/02/10 15:07:30 INFO mapreduce.JobSubmitter: number of splits:2
21/02/10 15:07:30 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1612976601004_0002
21/02/10 15:07:31 INFO impl.YarnClientImpl: Submitted application application_1612976601004_0002
21/02/10 15:07:31 INFO mapreduce.Job: The url to track the job: http://quickstart.cloudera:8088/proxy/application_1612976601004_0002/
21/02/10 15:07:31 INFO mapreduce.Job: Running job: job_1612976601004_0002
21/02/10 15:07:46 INFO mapreduce.Job: Job job_1612976601004_0002 running in uber mode : false
21/02/10 15:07:46 INFO mapreduce.Job:  map 0% reduce 0%
21/02/10 15:08:03 INFO mapreduce.Job:  map 50% reduce 0%
21/02/10 15:08:04 INFO mapreduce.Job:  map 100% reduce 0%
21/02/10 15:08:13 INFO mapreduce.Job:  map 100% reduce 100%
```

<hdfs commands>

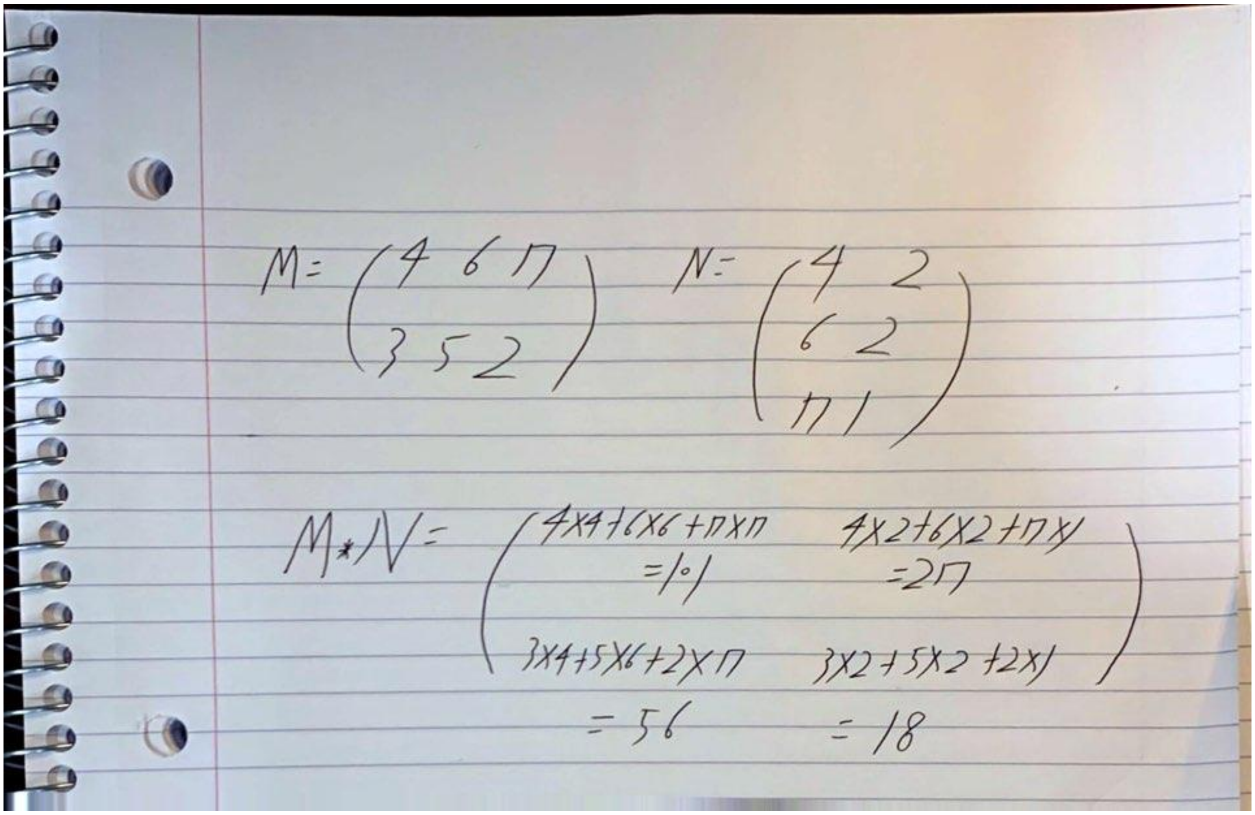


/ user / hadoop / output / part-r-00000

```
0,0,101.0
0,1,27.0
1,0,56.0
1,1,18.0
```

<OUTPUT>

We have a i x j matrix M, whose element in row i and column j will be denoted mij and a j x k matrix N whose element in row j and column k is donated by njk then the product P = MN will be i x k matrix P whose element in row i and column k will be donated by pik , where p(i,k) = mij * njk



$$M = \begin{pmatrix} 4 & 6 & 7 \\ 3 & 5 & 2 \end{pmatrix} \qquad N = \begin{pmatrix} 4 & 2 \\ 6 & 2 \\ 7 & 1 \end{pmatrix}$$

$$M * N = \begin{pmatrix} 4\times4+6\times6+7\times7 & 4\times2+6\times2+7\times1 \\ =101 & =27 \\ 3\times4+5\times6+2\times7 & 3\times2+5\times2+2\times1 \\ =56 & =18 \end{pmatrix}$$

Algorithm 1: The Map Function

For each element (mij) of matrix_M, it will create (key, value) pairs as ((i,k), (M,j,mij)) for k =1, 2 , ... up to the number of columns of N.

For each element (njk) of matrix_N, it will create (key, value) pairs as ((i,k), (N,j,njk)) for k =1, 2 , ... up to the number of columns of M.

Algorithm 2: The Reduce Function

```
For each key (i,k);
    it will sort values begin with M by j in the list List-M
    it will sort values begin with N by j in the list List-N
    it will multiply mij abd njk for the jth value of each list.
    it will sum up mij*njk
It will return the result.
```