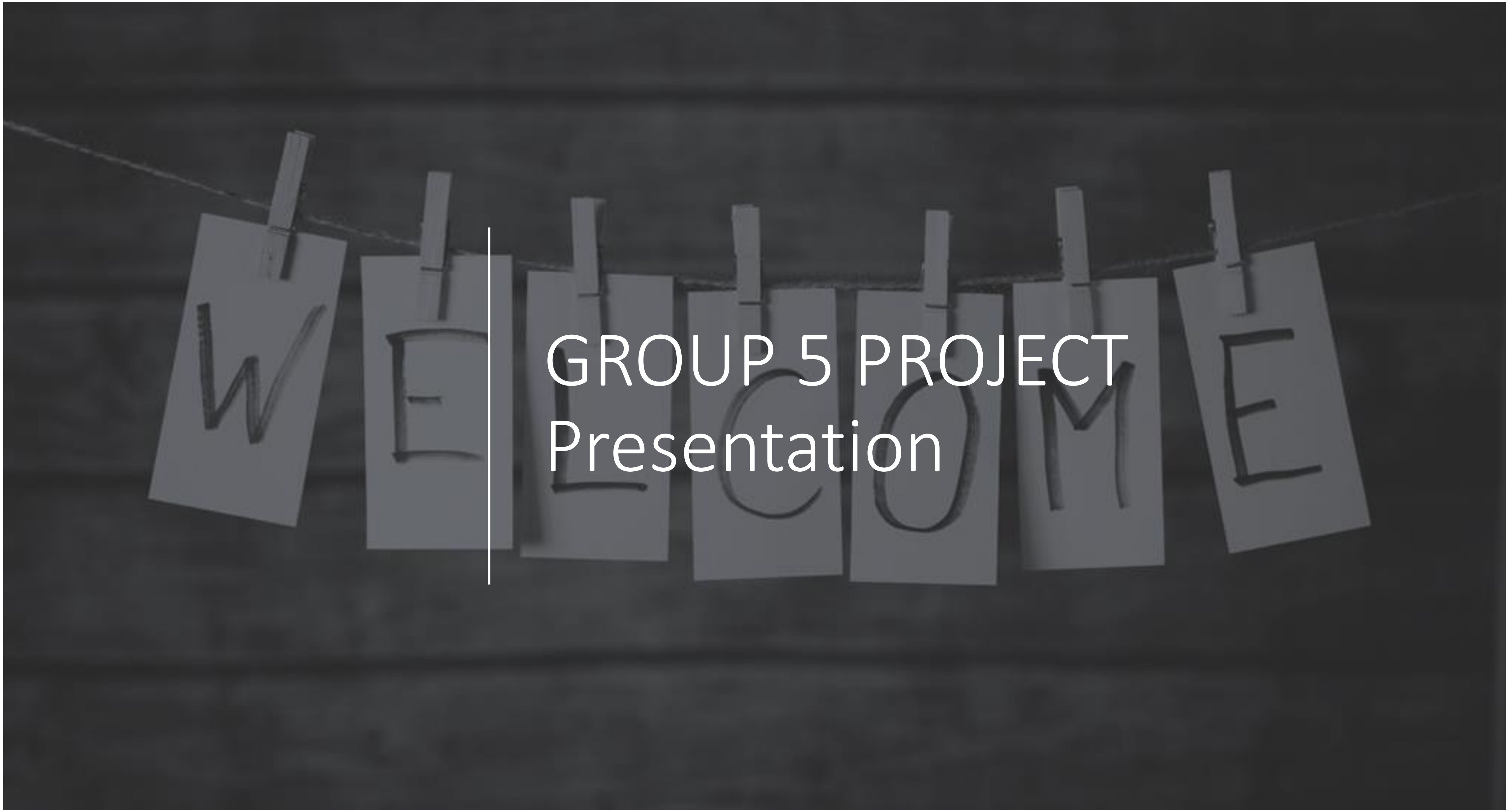
1871

UNIVERSITY OF CENTRAL MISSOURI

Big Data Analytics - 13418
W300 B - Fall 22



GROUP 5 PROJECT Presentation

PROJECT GUIDE

Dr. Mohammad Rawashdeh

Assistant Professor of
Computer Science
dept

MEET OUR TEAM !

Shrinika Telu

700741742

Durga Babu Padam

700726311

Mani Teja Mukka

700741007

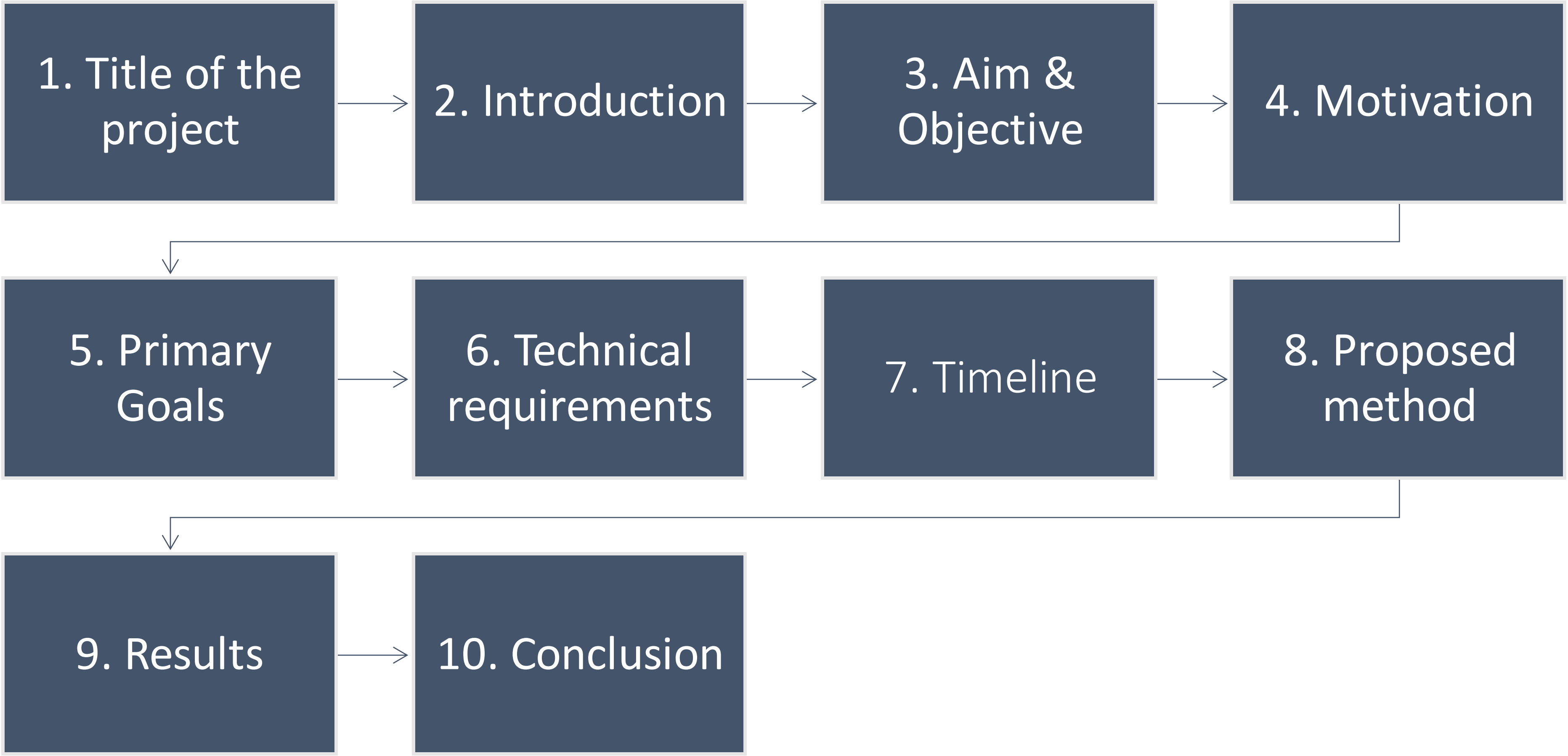
Praveen
Kumar Nutulapati

700742048

Srujana Maryada

700721987

Overview



1. Title of the project

2. Introduction

3. Aim & Objective

4. Motivation

5. Primary Goals

6. Technical requirements

7. Timeline

8. Proposed method

9. Results

10. Conclusion

MATRIX MULTIPLICATION

- MapReduce application

INTRODUCTION

Mapper is a function which process the input data. The mapper processes the data and creates several small chunks of data.

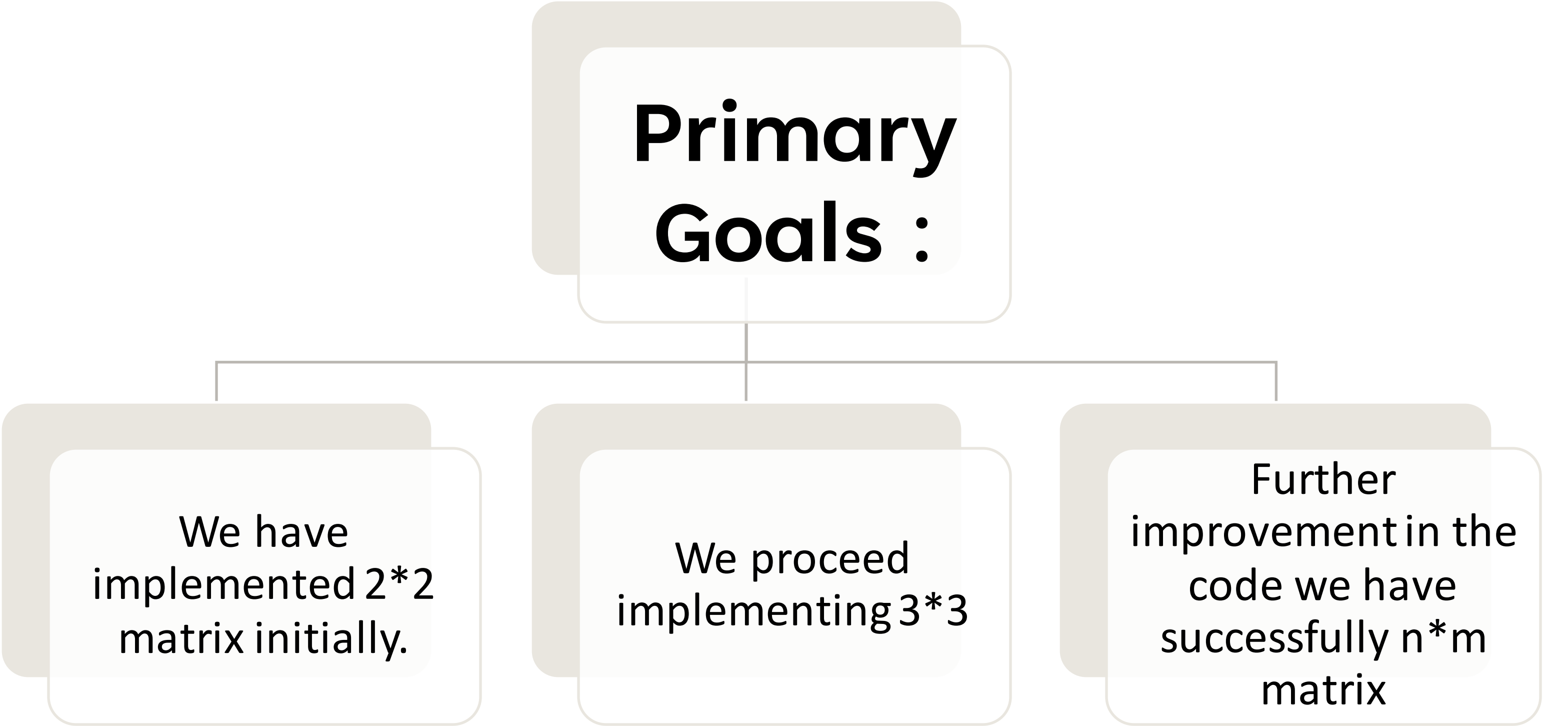
Reducer takes the output of the Mapper process each of them to generate the output.

Aim:

Aim is to Implement a MapReduce application to perform Matrix Multiplication.

Objective :

For Multiplication large scale data we are using MapReduce Code for matrix Multiplication in our project.



Primary Goals :

We have implemented 2*2 matrix initially.

We proceed implementing 3*3

Further improvement in the code we have successfully n*m matrix

Technical requirements

The technical requirements for this project are mentioned below:

a. Hardware Requirements

- PC with 2GB Ram
- Celeron Processor or Above

b. Software Requirements

- Visual studio code (platform independent)

TIMELINE

Week 1	Planning
Week 2	Designing
Week 3&4	Implementation of code
Week 5	Gathering Input and output sample

Proposed Methods for MapReduce

MapReduce is a technique in which a huge program is subdivided into small tasks and run parallelly to make computation faster, save time, and mostly used in distributed systems. It has 2 important parts:

Mapper: It takes raw data input and organizes into key, value pairs. For example, In a dictionary, you search for the word "Data" and its associated meaning is "facts and statistics collected together for reference or analysis". Here the Key is *Data* and the **Value** associated with is *facts and statistics collected together for reference or analysis.*

Reducer: It is responsible for processing data in parallel and produce final output.

MapReduce: Overview

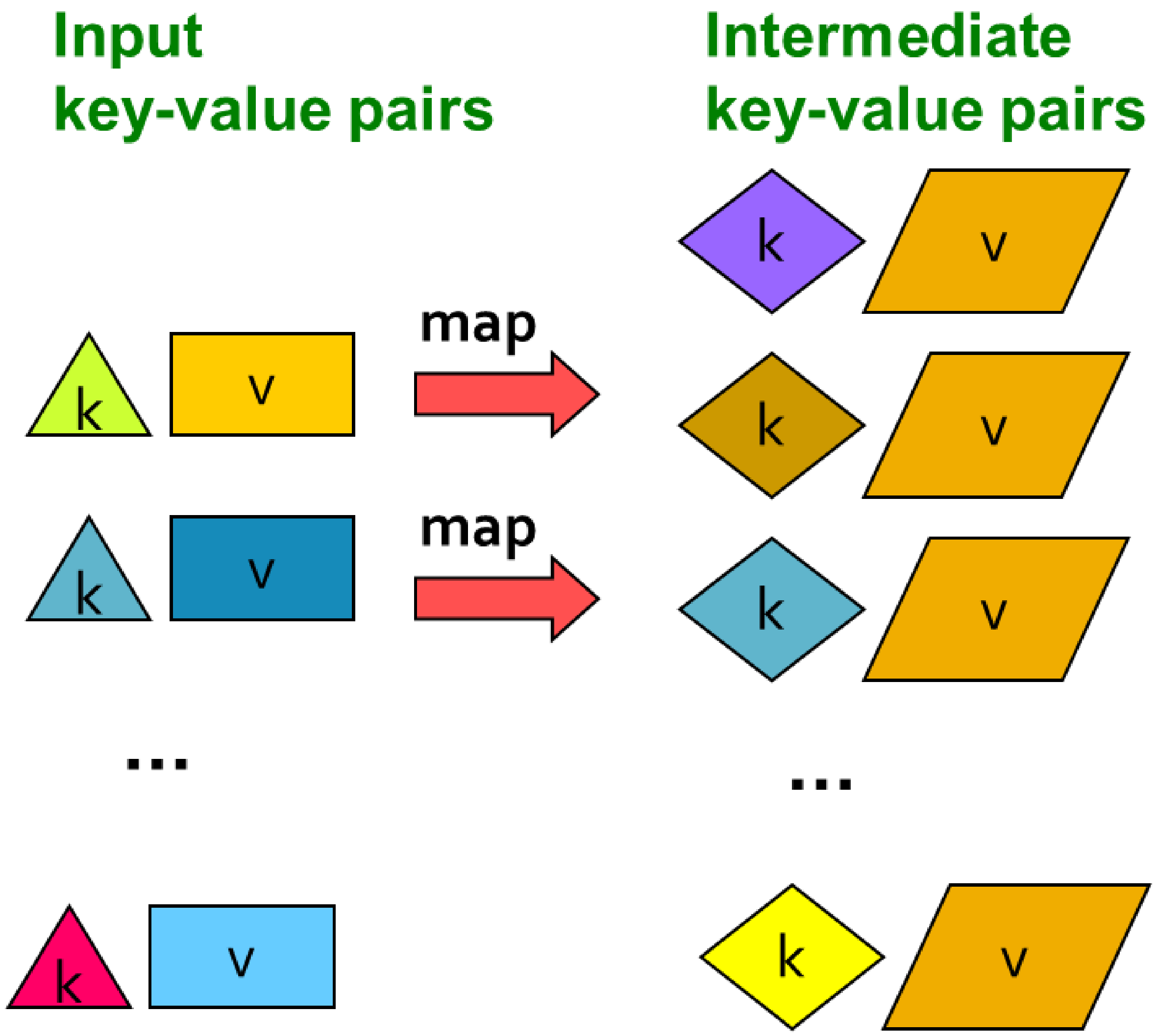
Sequentially read a lot of data

Map:
Extract something you care about from each record(keys).
Scan input file record-at-a-time

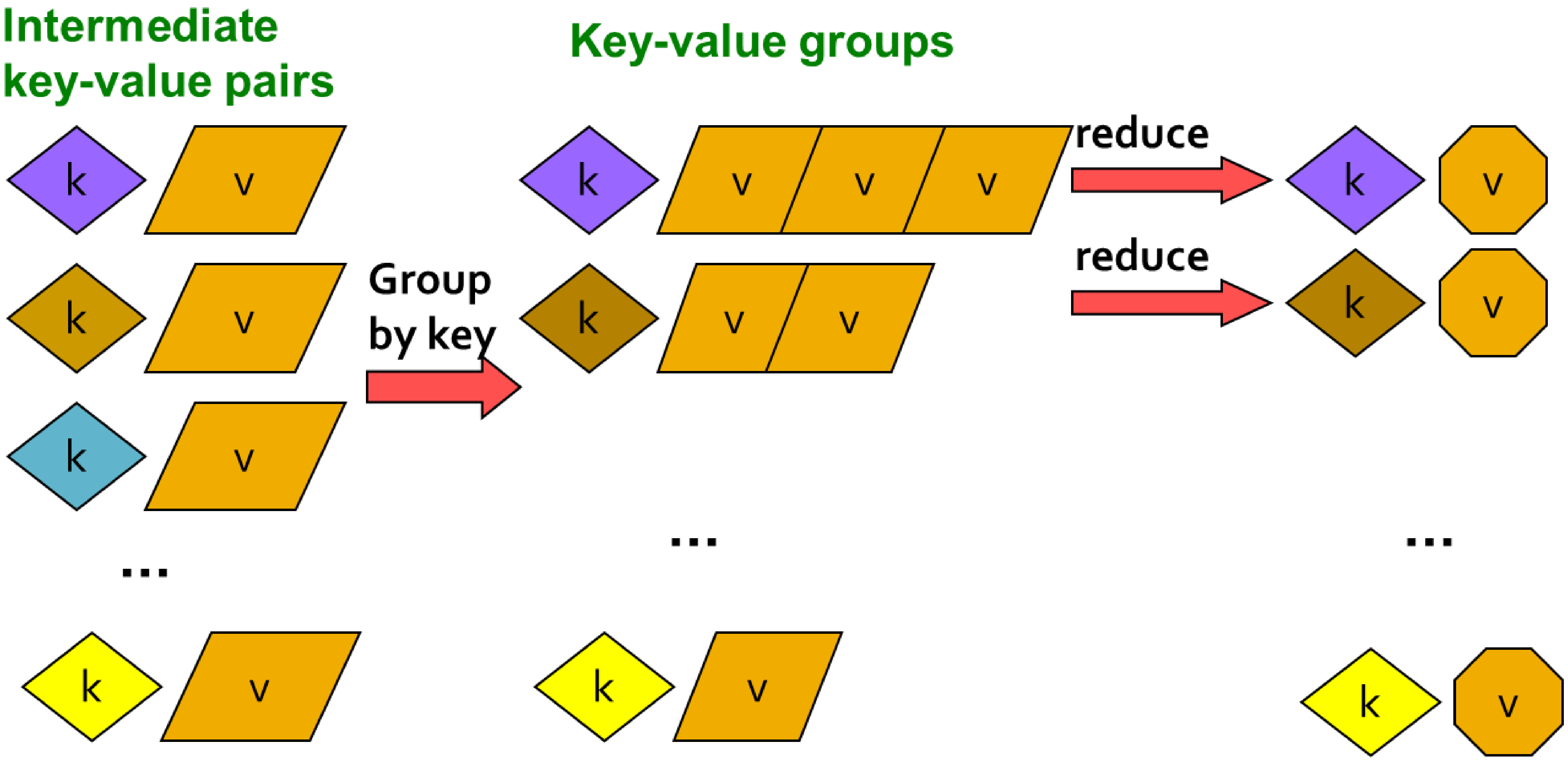
Group by key: Sort and Shuffle

Reduce:
Aggregate, summarize, filter or transform
Write the result

MapReduce: The Map Step



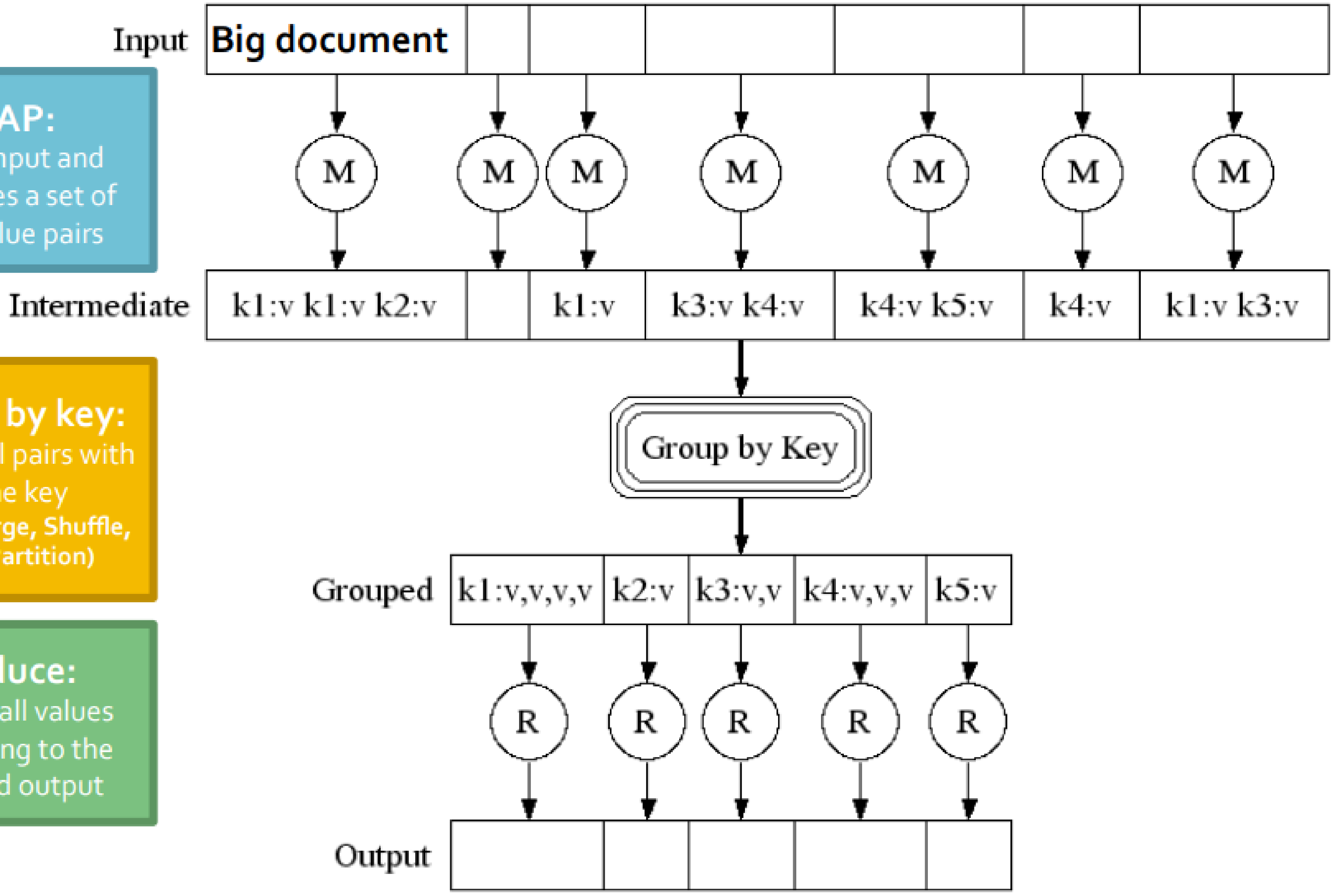
MapReduce: The Reduce Step



MAP:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key
(Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



Input: Big document

M M M M M M M

Intermediate: k1:v k1:v k2:v | | k1:v | k3:v k4:v | k4:v k5:v | k4:v | k1:v k3:v

Group by Key

Grouped: k1:v,v,v,v | k2:v | k3:v,v | k4:v,v,v | k5:v

R R R R R

Output

Map-Reduce environment takes care of :

1

Partitioning the input data

2

Scheduling the program’s execution across a set of machines

3

Performing the **group by key** step

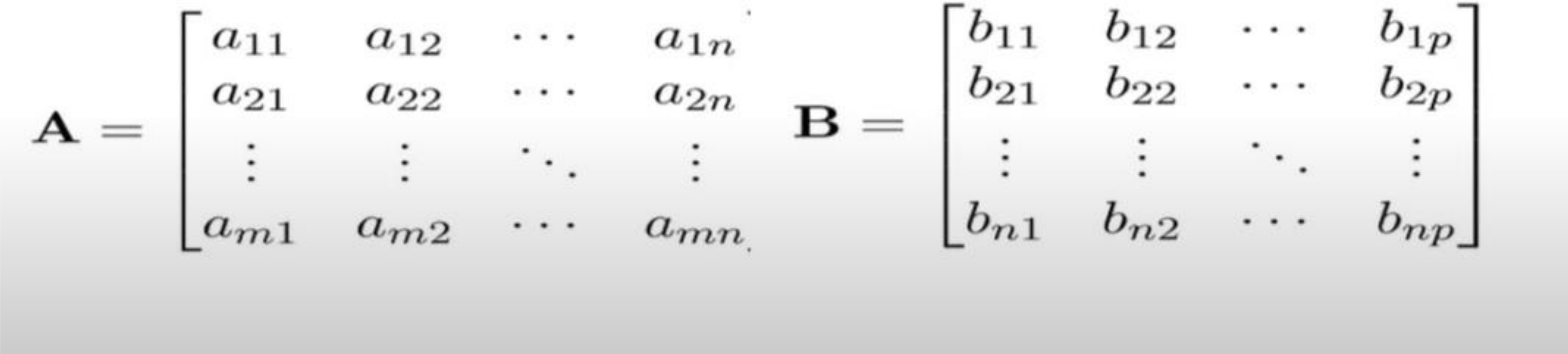
4

Handling machine failures

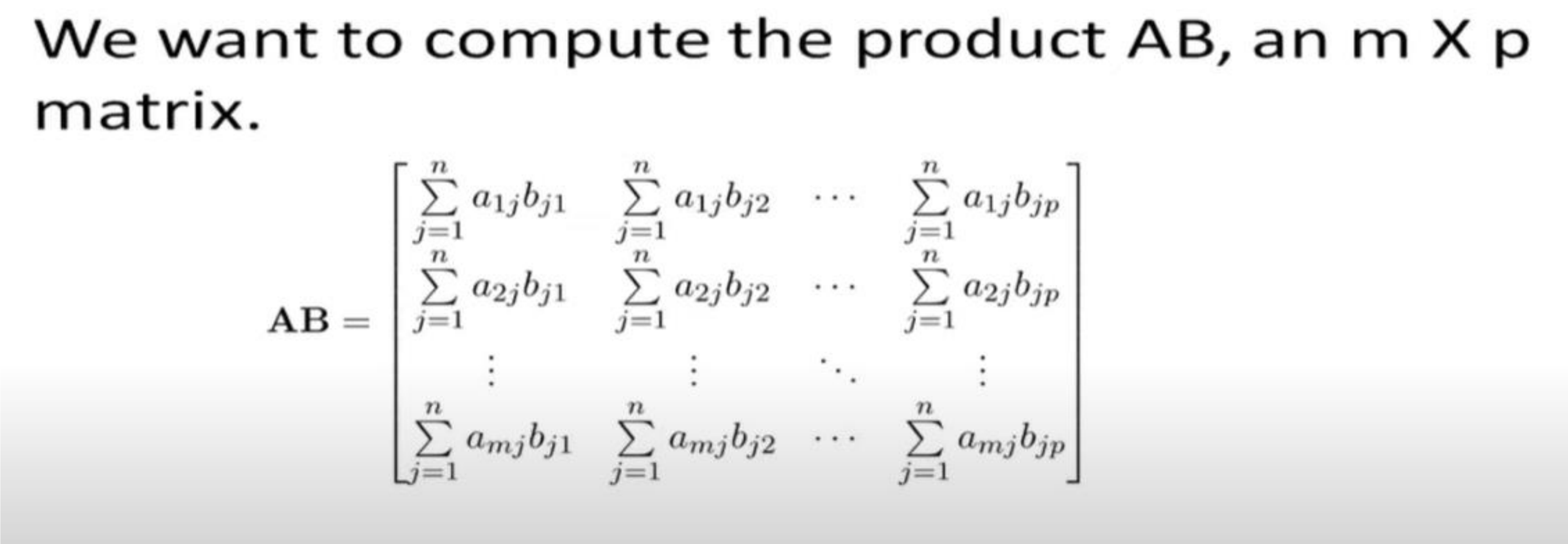
5

Managing required inter-machine communication

- Let A be an m X n matrix and B an n X p matrix.



$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix}$$



We want to compute the product AB, an m X p matrix.

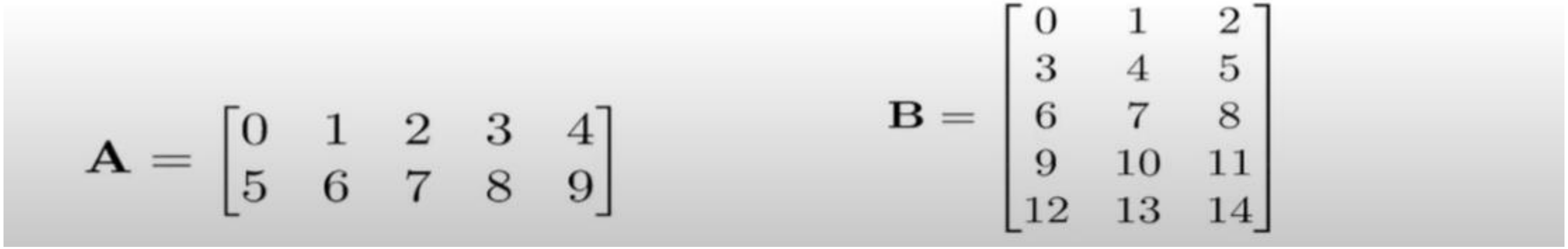
$$\mathbf{AB} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}b_{j1} & \sum_{j=1}^{n} a_{1j}b_{j2} & \cdots & \sum_{j=1}^{n} a_{1j}b_{jp} \\ \sum_{j=1}^{n} a_{2j}b_{j1} & \sum_{j=1}^{n} a_{2j}b_{j2} & \cdots & \sum_{j=1}^{n} a_{2j}b_{jp} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{n} a_{mj}b_{j1} & \sum_{j=1}^{n} a_{mj}b_{j2} & \cdots & \sum_{j=1}^{n} a_{mj}b_{jp} \end{bmatrix}$$

INPUT

The input file has one line of the following format for each non-zero element m_{ij} of a matrix M:

<M><i><j>< m_{ij} >

Suppose



$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \end{bmatrix}$$

Sample Input Collection

The input file that represents A and B has the following lines:

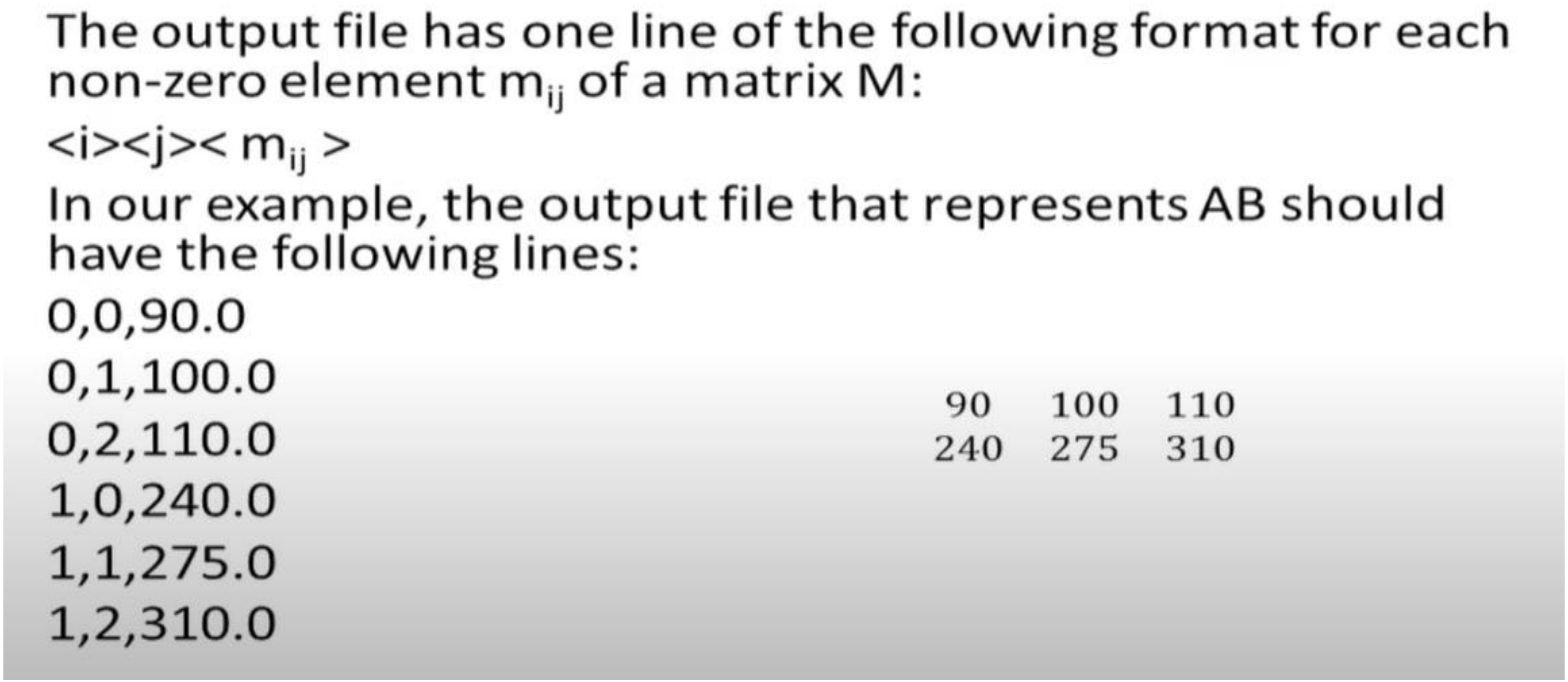
A, 0, 1, 1.0
A, 0, 2, 2.0
A, 0, 3, 3.0
A, 0, 4, 4.0
A, 1, 0, 5.0
A, 1, 1, 6.0
A, 1, 2, 7.0
A, 1, 3, 8.0
A, 1, 4, 9.0

B, 0, 1, 1.0
B, 0, 2, 2.0
B, 1, 0, 3.0
B, 1, 1, 4.0
B, 1, 2, 5.0
B, 2, 0, 6.0
B, 2, 1, 7.0
B, 3, 0, 9.0
B, 3, 1, 10.0
B, 3, 2, 11.0
B, 4, 0, 12.0
B, 4, 1, 13.0
B, 4, 2, 14.0

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 & 9 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \\ 9 & 10 & 11 \\ 12 & 13 & 14 \end{bmatrix}$$

Sample Output



The output file has one line of the following format for each non-zero element m_{ij} of a matrix M:

<i><j>< m_{ij} >

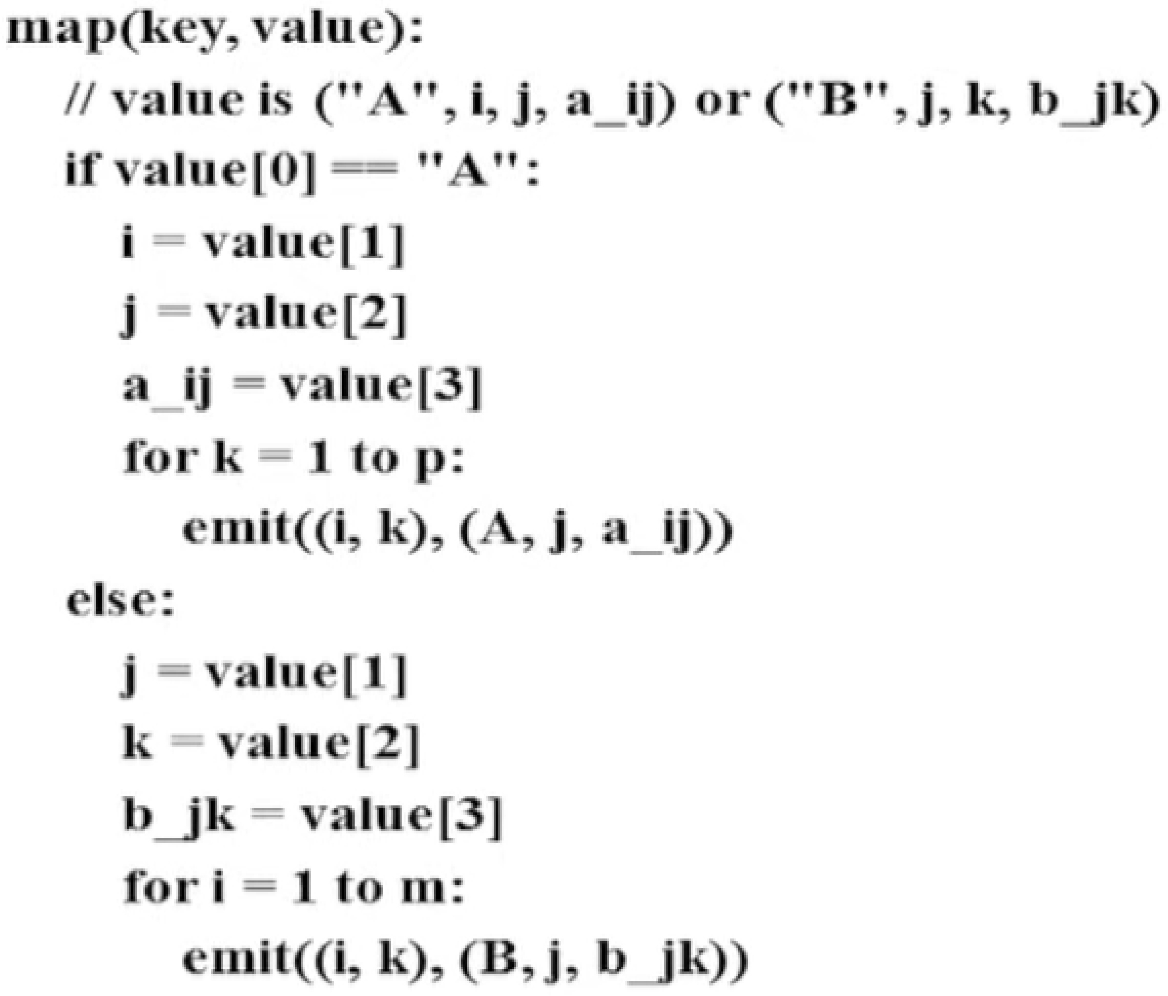
In our example, the output file that represents AB should have the following lines:

```
0,0,90.0
0,1,100.0
0,2,110.0
1,0,240.0
1,1,275.0
1,2,310.0
```

$$\begin{matrix} 90 & 100 & 110 \\ 240 & 275 & 310 \end{matrix}$$

PSEUDOCODE

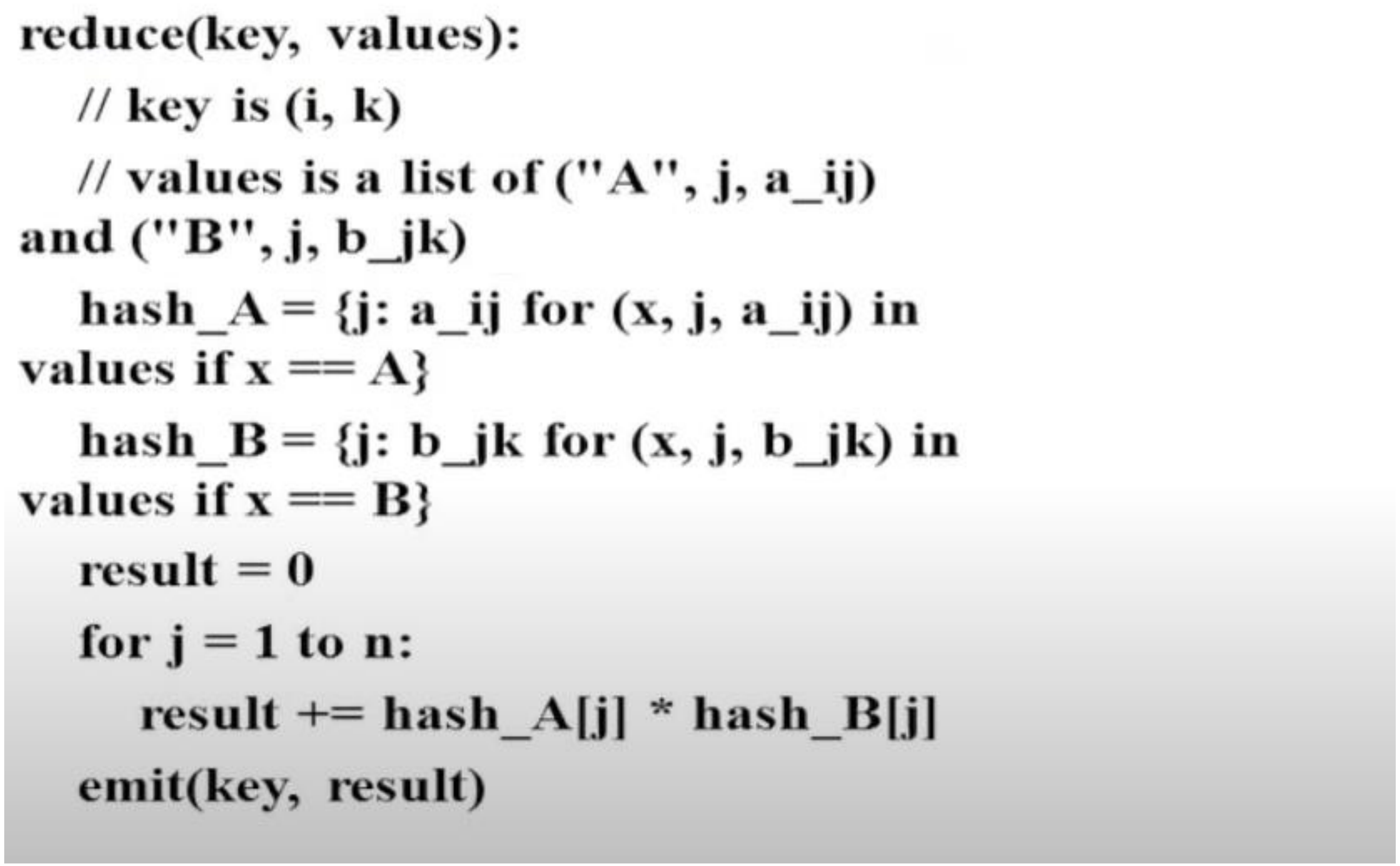
Mapper



```
map(key, value):
    // value is ("A", i, j, a_ij) or ("B", j, k, b_jk)
    if value[0] == "A":
        i = value[1]
        j = value[2]
        a_ij = value[3]
        for k = 1 to p:
            emit((i, k), (A, j, a_ij))
    else:
        j = value[1]
        k = value[2]
        b_jk = value[3]
        for i = 1 to m:
            emit((i, k), (B, j, b_jk))
```

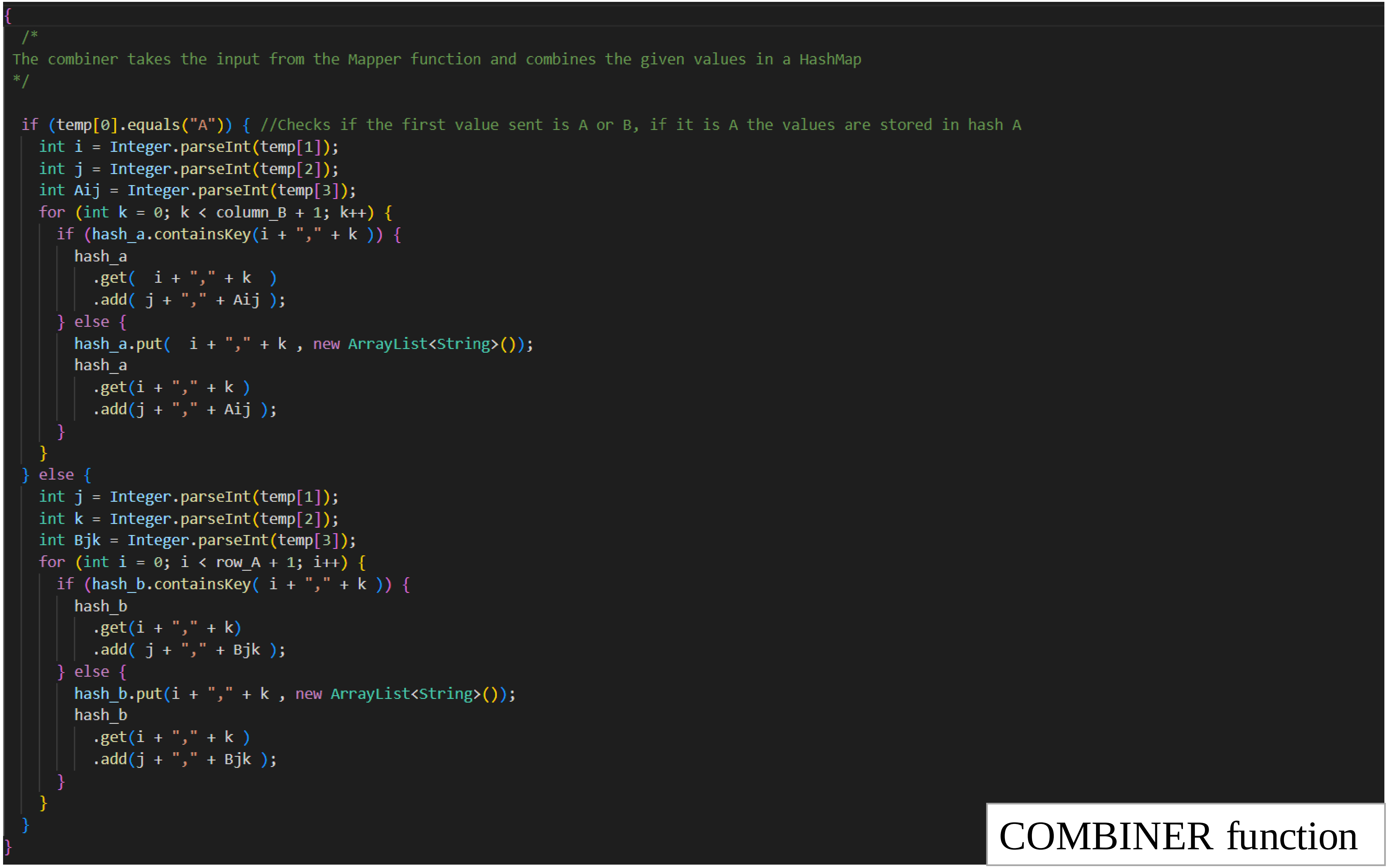
PSEUDOCODE

Reducer



```
reduce(key, values):
    // key is (i, k)
    // values is a list of ("A", j, a_ij) and ("B", j, b_jk)
    hash_A = {j: a_ij for (x, j, a_ij) in values if x == A}
    hash_B = {j: b_jk for (x, j, b_jk) in values if x == B}
    result = 0
    for j = 1 to n:
        result += hash_A[j] * hash_B[j]
    emit(key, result)
```

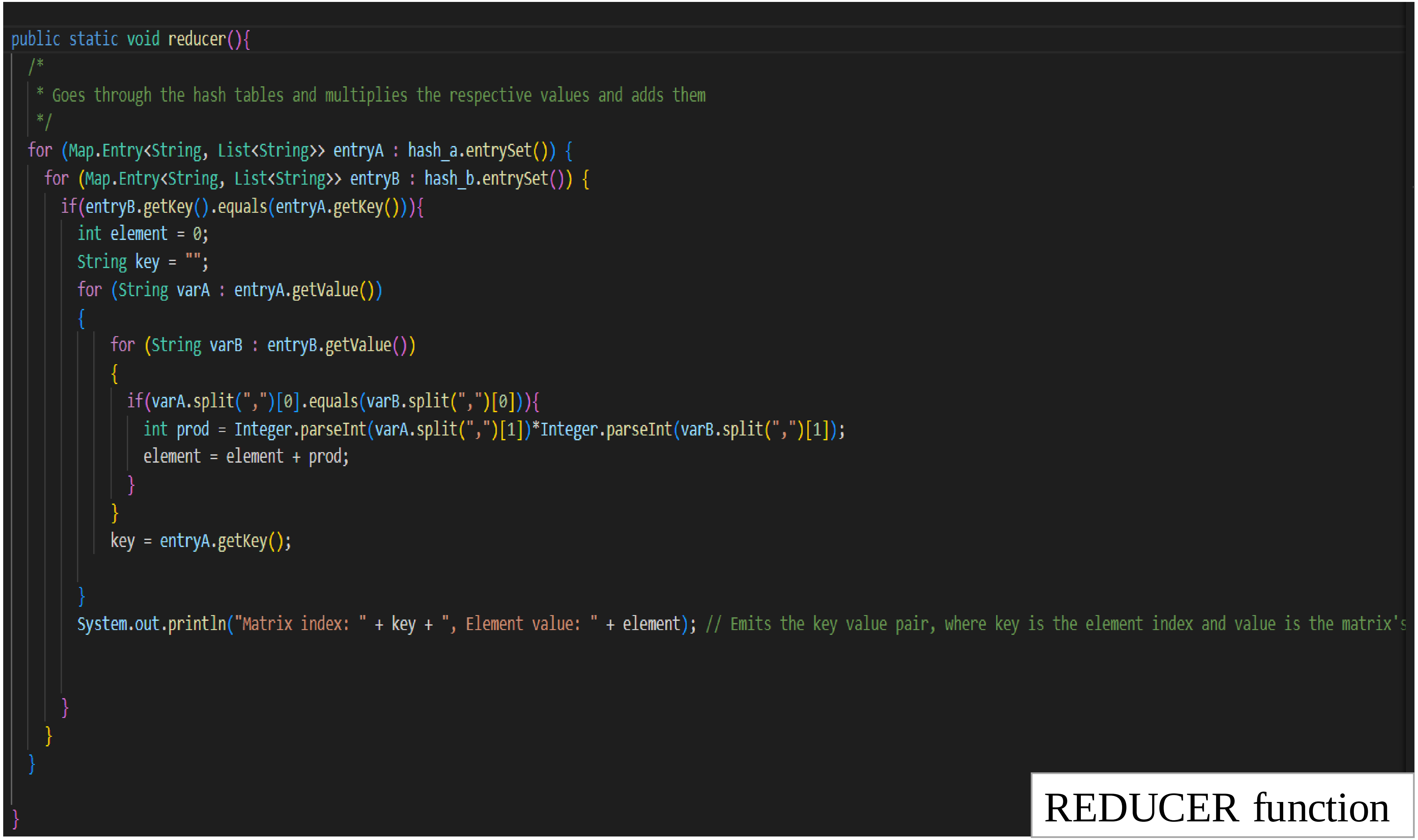
Code for Matrix Multiplication



```
{
 /*
The combiner takes the input from the Mapper function and combines the given values in a HashMap
*/

 if (temp[0].equals("A")) { //Checks if the first value sent is A or B, if it is A the values are stored in hash A
  int i = Integer.parseInt(temp[1]);
  int j = Integer.parseInt(temp[2]);
  int Aij = Integer.parseInt(temp[3]);
  for (int k = 0; k < column_B + 1; k++) {
   if (hash_a.containsKey(i + "," + k )) {
    hash_a
     .get(  i + "," + k  )
     .add( j + "," + Aij );
   } else {
    hash_a.put(  i + "," + k , new ArrayList<String>());
    hash_a
     .get(i + "," + k )
     .add(j + "," + Aij );
   }
  }
 } else {
  int j = Integer.parseInt(temp[1]);
  int k = Integer.parseInt(temp[2]);
  int Bjk = Integer.parseInt(temp[3]);
  for (int i = 0; i < row_A + 1; i++) {
   if (hash_b.containsKey( i + "," + k )) {
    hash_b
     .get(i + "," + k)
     .add( j + "," + Bjk );
   } else {
    hash_b.put(i + "," + k , new ArrayList<String>());
    hash_b
     .get(i + "," + k )
     .add(j + "," + Bjk );
   }
  }
 }
}
```

COMBINER function



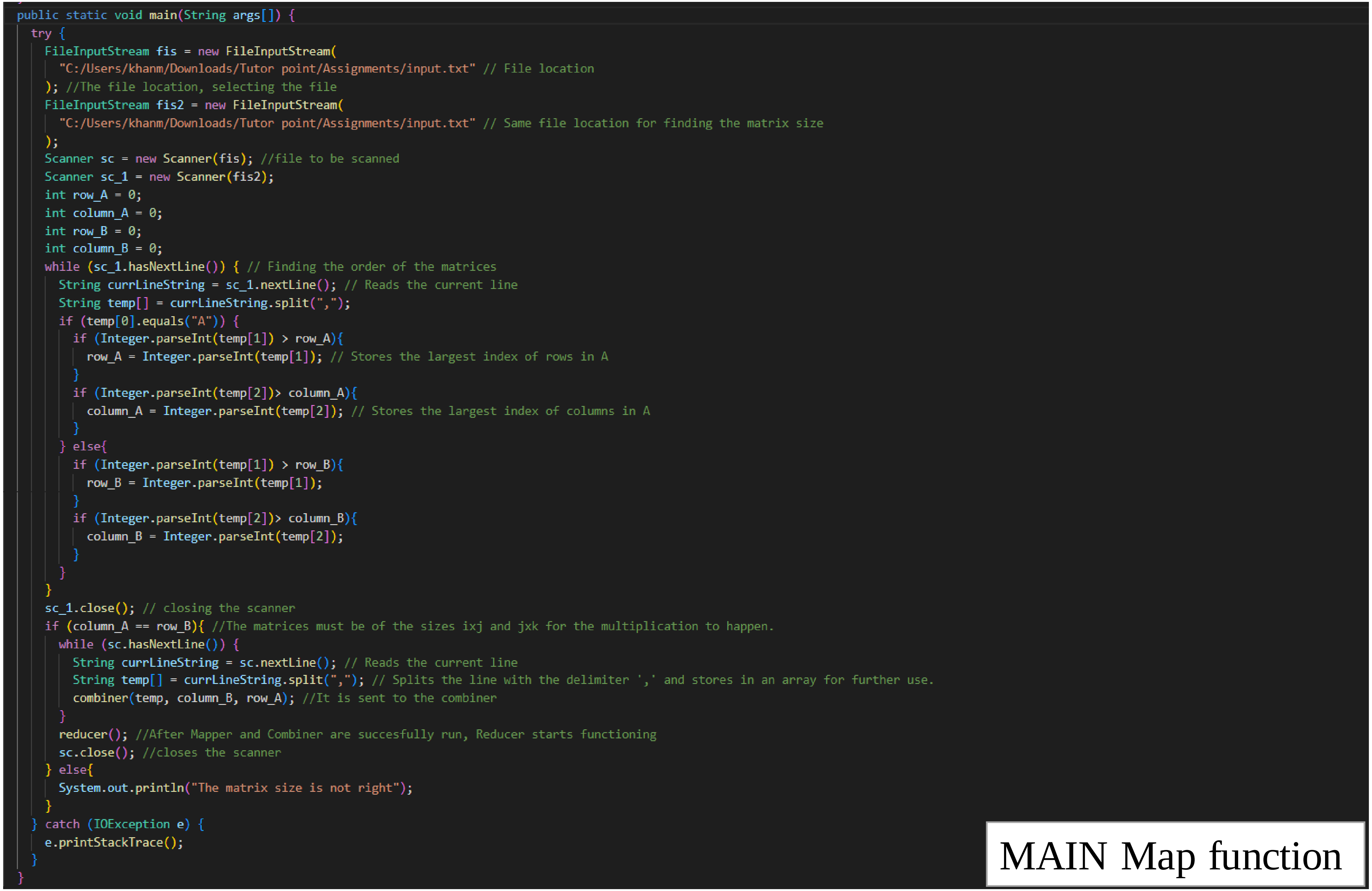
```java
public static void reducer(){
    /*
     * Goes through the hash tables and multiplies the respective values and adds them
     */
    for (Map.Entry<String, List<String>> entryA : hash_a.entrySet()) {
        for (Map.Entry<String, List<String>> entryB : hash_b.entrySet()) {
            if(entryB.getKey().equals(entryA.getKey())){
                int element = 0;
                String key = "";
                for (String varA : entryA.getValue())
                {
                    for (String varB : entryB.getValue())
                    {
                        if(varA.split(",")[0].equals(varB.split(",")[0])){
                            int prod = Integer.parseInt(varA.split(",")[1])*Integer.parseInt(varB.split(",")[1]);
                            element = element + prod;
                        }
                    }
                    key = entryA.getKey();

                }
                System.out.println("Matrix index: " + key + ", Element value: " + element); // Emits the key value pair, where key is the element index and value is the matrix's


            }
        }
    }

}
```

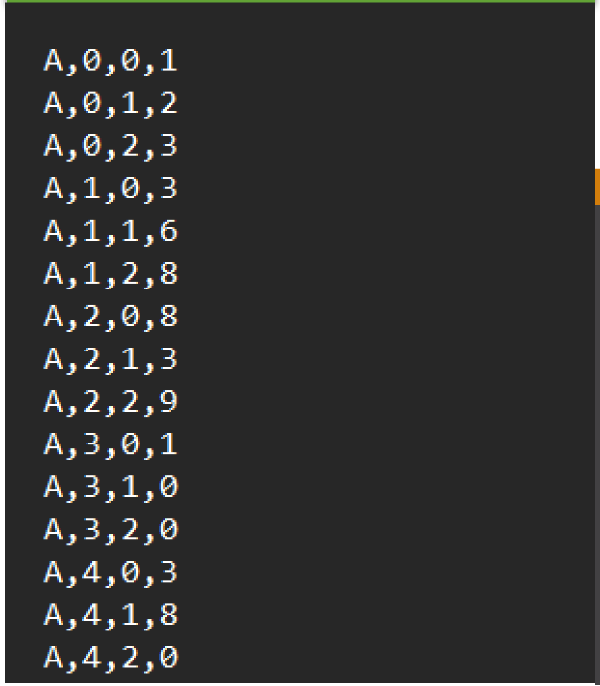
REDUCER function



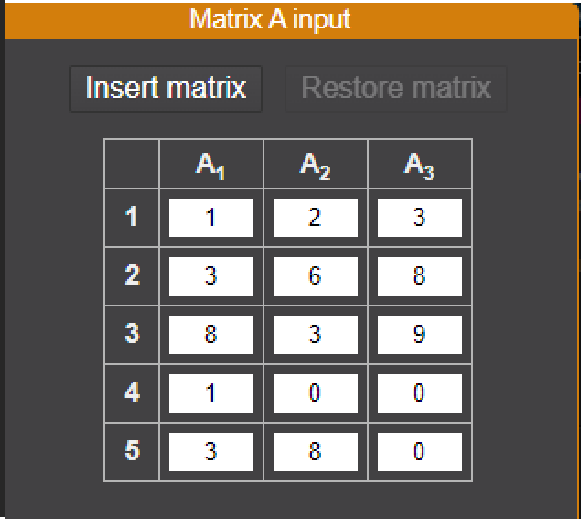
```java
public static void main(String args[]) {
  try {
    FileInputStream fis = new FileInputStream(
      "C:/Users/khanm/Downloads/Tutor point/Assignments/input.txt" // File location
    ); //The file location, selecting the file
    FileInputStream fis2 = new FileInputStream(
      "C:/Users/khanm/Downloads/Tutor point/Assignments/input.txt" // Same file location for finding the matrix size
    );
    Scanner sc = new Scanner(fis); //file to be scanned
    Scanner sc_1 = new Scanner(fis2);
    int row_A = 0;
    int column_A = 0;
    int row_B = 0;
    int column_B = 0;
    while (sc_1.hasNextLine()) { // Finding the order of the matrices
      String currLineString = sc_1.nextLine(); // Reads the current line
      String temp[] = currLineString.split(",");
      if (temp[0].equals("A")) {
        if (Integer.parseInt(temp[1]) > row_A){
          row_A = Integer.parseInt(temp[1]); // Stores the largest index of rows in A
        }
        if (Integer.parseInt(temp[2])> column_A){
          column_A = Integer.parseInt(temp[2]); // Stores the largest index of columns in A
        }
      } else{
        if (Integer.parseInt(temp[1]) > row_B){
          row_B = Integer.parseInt(temp[1]);
        }
        if (Integer.parseInt(temp[2])> column_B){
          column_B = Integer.parseInt(temp[2]);
        }
      }
    }
    sc_1.close(); // closing the scanner
    if (column_A == row_B){ //The matrices must be of the sizes ixj and jxk for the multiplication to happen.
      while (sc.hasNextLine()) {
        String currLineString = sc.nextLine(); // Reads the current line
        String temp[] = currLineString.split(","); // Splits the line with the delimiter ',' and stores in an array for further use.
        combiner(temp, column_B, row_A); //It is sent to the combiner
      }
      reducer(); //After Mapper and Combiner are succesfully run, Reducer starts functioning
      sc.close(); //closes the scanner
    } else{
      System.out.println("The matrix size is not right");
    }
  } catch (IOException e) {
    e.printStackTrace();
  }
}
```

MAIN Map function

MatrixA - (5 X 3)

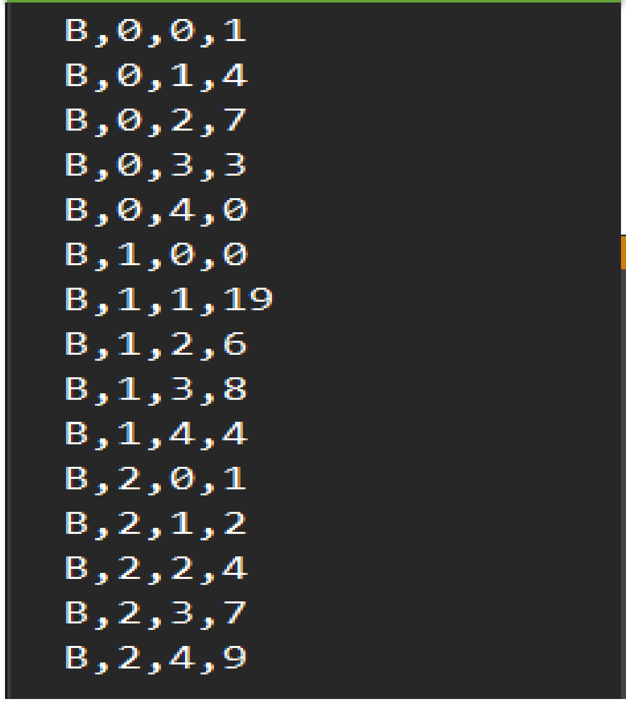


```
A,0,0,1
A,0,1,2
A,0,2,3
A,1,0,3
A,1,1,6
A,1,2,8
A,2,0,8
A,2,1,3
A,2,2,9
A,3,0,1
A,3,1,0
A,3,2,0
A,4,0,3
A,4,1,8
A,4,2,0
```

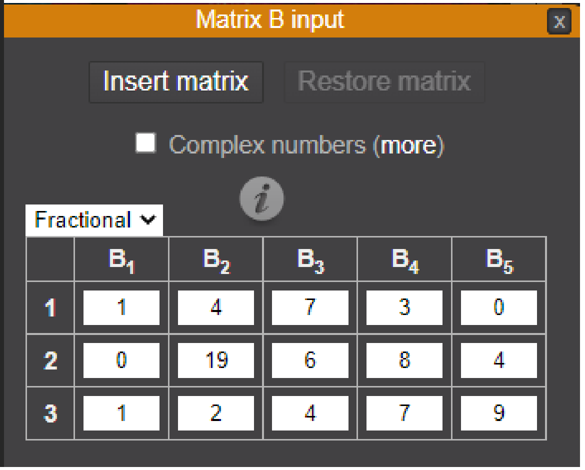


	A_1	A_2	A_3
1	1	2	3
2	3	6	8
3	8	3	9
4	1	0	0
5	3	8	0

MatrixB - (3 X 5)



```
B,0,0,1
B,0,1,4
B,0,2,7
B,0,3,3
B,0,4,0
B,1,0,0
B,1,1,19
B,1,2,6
B,1,3,8
B,1,4,4
B,2,0,1
B,2,1,2
B,2,2,4
B,2,3,7
B,2,4,9
```



☐ Complex numbers (more)

Fractional

	B_1	B_2	B_3	B_4	B_5
1	1	4	7	3	0
2	0	19	6	8	4
3	1	2	4	7	9

Matrix Multiplication Calculation

c11 = 1 x 1 + 2 x 0 + 3 x 1 = 4

c12 = 1 x 4 + 2 x 19 + 3 x 2 = 48

c13 = 1 x 7 + 2 x 6 + 3 x 4 = 31

c14 = 1 x 3 + 2 x 8 + 3 x 7 = 40

c15 = 1 x 0 + 2 x 4 + 3 x 9 = 35

c21 = 3 x 1 + 6 x 0 + 8 x 1 = 11

c22 = 3 x 4 + 6 x 19 + 8 x 2 = 142

c23 = 3 x 7 + 6 x 6 + 8 x 4 = 89

c24 = 3 x 3 + 6 x 8 + 8 x 7 = 113

c25 = 3 x 0 + 6 x 4 + 8 x 9 = 96

c31 = 8 x 1 + 3 x 0 + 9 x 1 = 17

c32 = 8 x 4 + 3 x 19 + 9 x 2 = 107

c33 = 8 x 7 + 3 x 6 + 9 x 4 = 110

c34 = 8 x 3 + 3 x 8 + 9 x 7 = 111

c35 = 8 x 0 + 3 x 4 + 9 x 9 = 93

c41 = 1 x 1 + 0 x 0 + 0 x 1 = 1

c42 = 1 x 4 + 0 x 19 + 0 x 2 = 4

c43 = 1 x 7 + 0 x 6 + 0 x 4 = 7

c44 = 1 x 3 + 0 x 8 + 0 x 7 = 3

c45 = 1 x 0 + 0 x 4 + 0 x 9 = 0

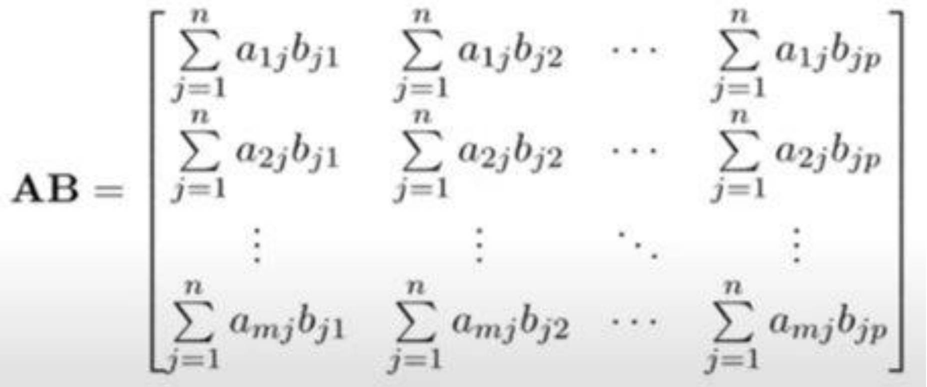
c51 = 3 x 1 + 8 x 0 + 0 x 1 = 3

c52 = 3 x 4 + 8 x 19 + 0 x 2 = 164

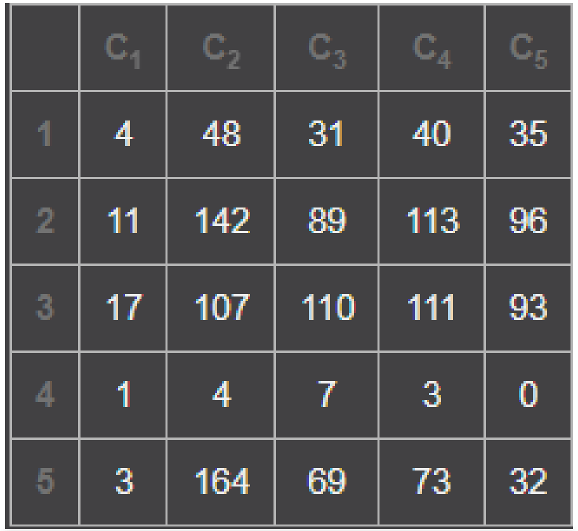
c53 = 3 x 7 + 8 x 6 + 0 x 4 = 69

c54 = 3 x 3 + 8 x 8 + 0 x 7 = 73

c55 = 3 x 0 + 8 x 4 + 0 x 9 = 32



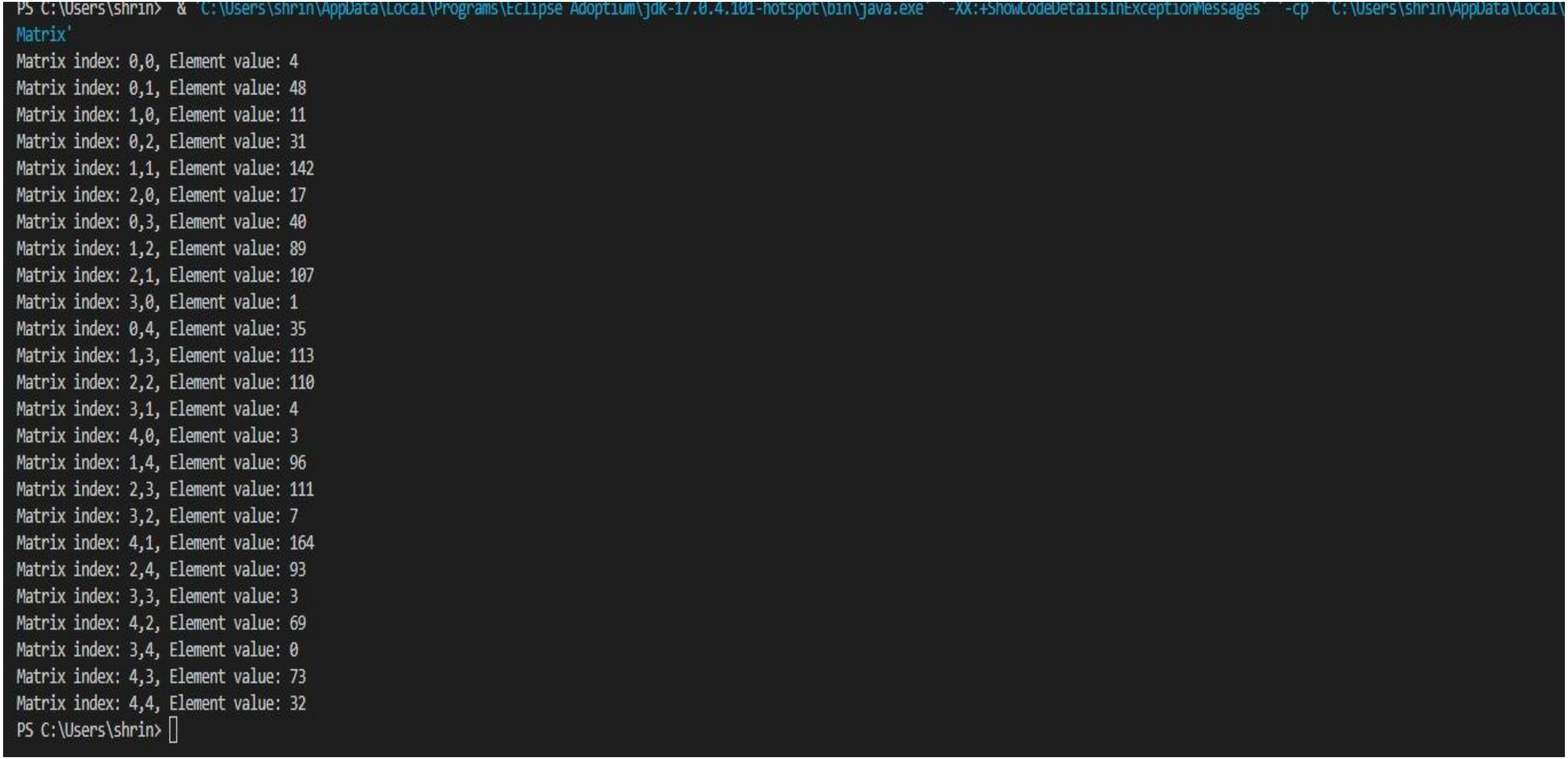
$$\mathbf{AB} = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}b_{j1} & \sum_{j=1}^{n} a_{1j}b_{j2} & \cdots & \sum_{j=1}^{n} a_{1j}b_{jp} \\ \sum_{j=1}^{n} a_{2j}b_{j1} & \sum_{j=1}^{n} a_{2j}b_{j2} & \cdots & \sum_{j=1}^{n} a_{2j}b_{jp} \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{j=1}^{n} a_{mj}b_{j1} & \sum_{j=1}^{n} a_{mj}b_{j2} & \cdots & \sum_{j=1}^{n} a_{mj}b_{jp} \end{bmatrix}$$



	C_1	C_2	C_3	C_4	C_5
1	4	48	31	40	35
2	11	142	89	113	96
3	17	107	110	111	93
4	1	4	7	3	0
5	3	164	69	73	32

Matrix Multiplication sample output

Output Matrix



```
PS C:\Users\shrin>  & 'C:\Users\shrin\AppData\Local\Programs\Eclipse Adoptium\jdk-17.0.4.101-hotspot\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\shrin\AppData\Local\
Matrix'
Matrix index: 0,0, Element value: 4
Matrix index: 0,1, Element value: 48
Matrix index: 1,0, Element value: 11
Matrix index: 0,2, Element value: 31
Matrix index: 1,1, Element value: 142
Matrix index: 2,0, Element value: 17
Matrix index: 0,3, Element value: 40
Matrix index: 1,2, Element value: 89
Matrix index: 2,1, Element value: 107
Matrix index: 3,0, Element value: 1
Matrix index: 0,4, Element value: 35
Matrix index: 1,3, Element value: 113
Matrix index: 2,2, Element value: 110
Matrix index: 3,1, Element value: 4
Matrix index: 4,0, Element value: 3
Matrix index: 1,4, Element value: 96
Matrix index: 2,3, Element value: 111
Matrix index: 3,2, Element value: 7
Matrix index: 4,1, Element value: 164
Matrix index: 2,4, Element value: 93
Matrix index: 3,3, Element value: 3
Matrix index: 4,2, Element value: 69
Matrix index: 3,4, Element value: 0
Matrix index: 4,3, Element value: 73
Matrix index: 4,4, Element value: 32
PS C:\Users\shrin> 
```

Product of Matrix A and Matrix B (AB)

	0	**1**	**2**	**3**	**4**
0	4	48	31	40	35
1	11	142	89	113	96
2	17	107	110	111	93
3	1	4	7	3	0
4	3	164	69	73	32

SUMMARY

Input: a set of key-value pairs

I.e. MATRIX A and MATRIX B filenames and their respective values

In Program, it specifies two methods:

Map(k, v) ® <k', v'>*

Takes a key-value pair and outputs a set of key-value pairs

E.g., key is the filename, value is a single line in the file

There is one Map call for every *(k,v)* pair

Reduce(k', <v'>*) ® <k', v''>*

All values *v'* with same key *k'* are reduced together and processed in *v'* order

There is one Reduce function call per unique key *k'*

Thus, Product of Matrix a and Matrix B is obtained as output using MapReduce Application.

Thank you