Lab 3 Report

Shane House (749524), Benjamin Rosen (858324)

School of Electrical & Information Engineering, University of the Witwatersrand, Private Bag 3, 2050, Johannesburg, South Africa

I. Introduction

The purpose of this lab was to learn to use map reduce frameworks to create two algorithms that perform matrix multiplication. Once the algorithms were created one was to be used to find nodes connected by length 3 in an unweighted directed graph which is represented as a matrix. The Python library Mrjob was selected to implement the algorithms. Section II below discusses the created algorithms and how they were implemented. Section III details how the node graph problem was solved, while Section IV gives the results of testing the algorithms and finally, Section V concludes the report.

II. Algorithms

A. Algorithm A

This algorithm has 2 stages. In the first map stage, the script reads in each element of the two matrices and returns the columns of the first matrix or the rows of the second matrix as the key and the position, value and matrix number as the value.

In the second stage, the algorithm returns the final positions in the output matrix as the key and the multiplied values as the value.

The reduce stage then adds all the mapped values together and prints out the position and value of each element in the output matrix.

Algorithm 1 Pseudo code for Algorithm A

```
For matrices A(i,j) and B(j,k)
In the map phase:
if matrix A
  yield key(j) value(row,column,value,1)
else
  yield key(j) value(row,column,value,2)

in the second stage:
create array matrix1 a fill it using values whose last element is 1
create array matrix2 and fill it using values whose last element is 2
for each entry i in matrix1
  for each entry j in matrix2
    yield key(i[row],j[column]) value(i[value]*j[value])

In reduce stage:
yield key[0] key[1] sum(values)
```

B. Algorithm B

In the mapping phase of this algorithm, the script reads in each element of the two input matrices and constructs a 2D array representation of them. It then groups the row and column needed to construct each element of the output matrix and yields them to be used in the reduction phase.

The reduction phase takes the above mentioned groups and multiplies the necessary values together and then adds the answers to get the final value for each element. It then prints the position and value to the output file.

Algorithm 2 Pseudo code for Algorithm B

```
For matrices A(i,j) and B(j,k)
In the map phase:
for each column k in B
    for each row i in A
        for each column j in A
            yield key(i,k) value(A(i,j))

for each row i in A
    for each row j in B
        for each column k in B
            yield key(i,k) value(B(i,j))

In reduce stage:
for each column j in A
    temp = values[j]*values[num_of_cols_in_A+j]
    sum += temp
yield key[0] key[1] sum
```

III. Graph Path Length Implementation

In order for this problem to be solved, first the node graph must be represented as a matrix. This is achieved by allotting each row to a node (1-N nodes). The same is done for the columns. A connection between nodes is represented as a number in the position *(i, j),* where *i* and *j* are the indices for the nodes that are connected. Since, for this lab, the graph is unweighted, the value of each position is either a 1 (if they are connected) or 0 (if they are unconnected). The fact that the lab specifies a directed graph implies that a filled position *(i, j)* (i.e. that *i* has a connection pointing to *j*) does not imply the converse *(j, i)*.

After this is done algorithm a can be used 3 times in succession to perform matrix multiplication of the matrix with itself to find the nodes that are of length 3, the number of these connections can then be mapped in the mapping phase of a map reduce framework and the reduce phase will count the number of times this occurs. This uses the fact that for a matrix *A*, representing a graph G, A^k will yield the number of connections between nodes that have a length k.

IV. Results

TABLE I
Results for Algorithm A and B

Algorithm	Matrix 1 Size	Matrix 2 Size	Time (s)
A	4x2	4x4	0.75521749
	200x100	500x200	101.32513048
	100x100 (outA1)	100x100 (outB1)	N/A
	1000x500 (outA2)	500x1000 (outB2)	N/A
	1000x500 (outA3)	500x1 (outB3)	N/A
B	4x2	4x4	0.34414595
	200x100	500x200	540.31728145
	100x100 (outA1)	100x100 (outB1)	N/A
	1000x500 (outA2)	500x1000 (outB2)	N/A
	1000x500 (outA3)	500x1 (outB3)	N/A

The results for the various input files are shown in Table I. Test files given by the lab is denoted by specifying the file name in brackets under the matrix size. As is shown, the test files did not run correctly. On inspecting the files, it was found that the input file for the second matrix was not complete. In addition, the graph matrix was given in upper echelon form, which is a suitable representation for an undirected graph but not for the lab-specified directed graph as mentioned in Section III.

Nevertheless, testing was successful with the authors' files. Although algorithm B has only one stage and algorithm A has two stages, algorithm A was found to preform much better for large matrices, whereas B was slightly faster for smaller data sets.

V. Conclusions

The map reduce framework can be used to solve matrix multiplication in one or two stages. Although the one stage algorithm out preforms the two stage algorithm for small matrices, the two stage algorithm is significantly faster for 2 stage algorithms. The created algorithms can also be used to solve node graph lengths when the node graph is represented as a matrix. The given lab files are not in the correct format/missing data points to be properly used for testing the speed of the algorithms.