Here are the results after computing G in MatLab. You can verify G is correct by comparing G to the assignment.

E =

     0     1     1     0     0     0

     0     0     1     0     1     0

     1     0     0     0     0     0

     1     0     1     0     0     1

     0     0     0     1     0     1

     0     0     0     0     0     0

E\_squared =

     1     0     1     0     1     0

     1     0     0     1     0     1

     0     1     1     0     0     0

     1     1     1     0     0     0

     1     0     1     0     0     1

     0     0     0     0     0     0

E\_cubed =

     1     1     1     1     0     1

     1     1     2     0     0     1

     1     0     1     0     1     0

     1     1     2     0     1     0

     1     1     1     0     0     0

     0     0     0     0     0     0

G =

     2     2     3     1     1     1

     2     1     3     1     1     2

     2     1     2     0     1     0

     3     2     4     0     1     1

     2     1     2     1     0     2

     0     0     0     0     0     0

Notice in the Row-Column multiplication rule that you are adding all the vector multiplication results together to form (AB)ij  = (ain)(bnj) + (ain)(bnj) where n >= 0.

(I would subscript i and j if I knew how, just remember that i and j will always refer to subscripts)

Since we are ignoring zero’s we are only adding matrix results > 0.

So after adding to our E list we should only have the following coordinates in our sparse list:

Coordinates = (Rowi, Columnj)

E = [(1,2), (1,3), (2,3), (2,5), (3,1), (4,1), (4,3), (4,6), (5,4), (5,6)]

Notice that the j value of a is equal to the i value of b:   a=(i,n) \* b(n,j) =  ( a == i, b == j ) ->   a = (1,2)  \*  b = (2,5) =  (1,5)

So step 1:

Gather all combinations of the form ain \* bnj = (ab)ij

From the E list, the resulting combinations are:

ain \* bnj = (ab)ij

(1,2) \* (2,3) = (1,3)

(1,2) \* (2,5) = (1,5)

(1,3) \* (3,1) = (1,1)

(2,3) \* (3,1) = (2,1)

(2,5) \* (5,4) = (2,4)

(2,5) \* (5,6) = (2,6)

(3,1) \* (1,2) = (3,2)

(3,1) \* (1,3) = (3,3)

(4,1) \* (1,2) = (4,2)

(For reference so you don’t have to scroll up

  E = [(1,2), (1,3), (2,3), (2,5), (3,1), (4,1), (4,3), (4,6), (5,4), (5,6)] )

(4,1) \* (1,3) = (4,3)

(4,3) \* (3,1) = (4,1)

(4,6) \* there is no value for which i == 6

(5,4) \* (4,1) = (5,1)

(5,4) \* (4,3) = (5,3)

(5,4) \* (4,6) = (5,6)

(5,6) \* there is no value for which i == 6

Now when these combinations are formed into an unsorted list we have:

Unsorted:

E\_squared = [(1,3), (1,5), (1,1), (2,1), (2,4), (2,6), (3,2), (3,3), (4,2), (4,3), (4,1), (5,1), (5,3), (5,6)]

Sorted:

E\_squared = [(1,1), (1,3), (1,5), (2,1), (2,4), (2,6), (3,2), (3,3), (4,1), (4,2), (4,3), (5,1), (5,3), (5,6)]

Compare this list of coordinates to the matrix E\_squared:

E\_squared =

     1     0     1     0     1     0

     1     0     0     1     0     1

     0     1     1     0     0     0

     1     1     1     0     0     0

     1     0     1     0     0     1

     0     0     0     0     0     0

Step 2 would be to add the resulting coordinates from ain\*bnj together to get the correct number of paths represented in the matrix. For instance if there was a combination of coordinates:

(1,2) \* (2,1) = (1,1)    and  (1,3) \* (3,1) = (1,1) so that E = [(1,1),(1,1)], the number of combinations to place in coordinate (Row1,Column1) = 2 from counting the number of combinations equal to (1,1).

You can see this from calculating E\*E\_squared like so:

Step 1:

               E = [(1,2), (1,3), (2,3), (2,5), (3,1), (4,1), (4,3), (4,6), (5,4), (5,6)] \*

E\_squared = [(1,1), (1,3), (1,5), (2,1), (2,4), (2,6), (3,2), (3,3), (4,1), (4,2), (4,3), (5,1), (5,3), (5,6)] =

(1,2) \*(2,1) = (1,1)

(1,2) \* (2,4) = (1,4)

(1,2) \* (2,6) = (1,6)

(1,3) \* (3,2) = (1,2)

(1,3) \* (3,3) = (1,3)

(2,3) \* (3,2) = (2,2)

(2,3) \* (3,3) = (2,3)

(2,5) \* (5,1) = (2,1)

(2,5) \* (5,3) = (2,3)

(2,5) \* (5,6) = (2,6)

(3,1) \* (1,1) = (3,1)

(For reference: E = [(1,2), (1,3), (2,3), (2,5), (3,1), (4,1), (4,3), (4,6), (5,4), (5,6)] \*

          E\_squared = [(1,1), (1,3), (1,5), (2,1), (2,4), (2,6), (3,2), (3,3), (4,1), (4,2), (4,3), (5,1), (5,3), (5,6)] =)

(3,1) \* (1,3) = (3,3)

(3,1) \* (1,5) = (3,5)

(4,1) \* (1,1) = (4,1)

(4,1) \* (1,3) = (4,3)

(4,1) \* (1,5) = (4,5)

(4,3) \* (3,2) = (4,2)

(4,3) \* (3,3) = (4,3)

(4,6) \* there is no i == 6

(5,4) \* (4,1) = (5,1)

(5,4) \* (4,2) = (5,2)

(5,4) \* (4,3) = (5,3)

(5,6) \* there is no i == 6

Unsorted:

E\_cubed = [(1,1), (1,4), (1,6), (1,2), (1,3), (2,2), (2,3), (2,1), (2,3), (2,6), (3,1), (3,3), (3,5), (4,1), (4,3), (4,5),

                   (4,2), (4,3), (5,1), (5,2), (5,3)]

Sorted:

E\_cubed = [(1,1), (1,2), (1,3), (1,4), (1,6), (2,1), (2,2), (2,3), (2,3), (2,6), (3,1), (3,3), (3,5), (4,1), (4,2), (4,3),

                   (4,3), (4,5),  (5,1), (5,2), (5,3)]

Verifying this list with E\_cubed from MatLab:

E\_cubed =

     1     1     1     1     0     1

     1     1     2     0     0     1

     1     0     1     0     1     0

     1     1     2     0     1     0

     1     1     1     0     0     0

     0     0     0     0     0     0

Notice that multiple combinations of the same coordinate == 2.

Now for computing G= E + E\_squared+E\_cubed all you have to do is count similar combinations (or add together stored values, but the homework says we don’t have to store values)

G = {E = [(1,2), (1,3), (2,3), (2,5), (3,1), (4,1), (4,3), (4,6), (5,4), (5,6)] +

        E\_squared = [(1,1), (1,3), (1,5), (2,1), (2,4), (2,6), (3,2), (3,3), (4,1), (4,2), (4,3), (5,1), (5,3), (5,6)] +

        E\_cubed = [(1,1), (1,2), (1,3), (1,4), (1,6), (2,1), (2,2), (2,3), (2,3),(2,6),(3,1),(3,3),(3,5),(4,1),(4,2),(4,3),

                             (4,3), (4,5),  (5,1), (5,2), (5,3)]}

G = [(1,1), (1,1), (1,2), (1,2), (1,3), (1,3), (1,3), (1,4), (1,5), (1,6), (2,1), (2,1), (2,2), (2,3), (2,3), (2,3), (2,4),

        (2,5), (2,6), (2,6), (3,1), (3,1), (3,2), (3,3), (3,3), (3,5), (4,1), (4,1), (4,1), (4,2), (4,2), (4,3), (4,3), (4,3),

        (4,3), (4,5), (4,6), (5,1), (5,1), (5,2), (5,3), (5,3), (5,4), (5,6), (5,6)]

Verifying this with G computed in MatLab and you can verify from the homework example:

G =

     2     2     3     1     1     1

     2     1     3     1     1     2

     2     1     2     0     1     0

     3     2     4     0     1     1

     2     1     2     1     0     2

     0     0     0     0     0     0

If you draw a map of all the vectors using the E matrix you will notice that:

i == all outdegree edges

j == all indegree edges

E == all outdegree and indegree edges of path length 1

E\_squared == all outdegree and indegree edges of path length 2 (this includes paths that travel from

                         A -> B , B->A; which is not a loop, it’s to outdegree paths that happen to return to point A.

                          A loop would be from A -> A, a single outdegree from A directed to A.)

E\_cubed == all outdegree and indegree edges of path length 3

So in order to calculate the top outdegree vertex you would add all the coordinates in a row and compare the result with the addition results from the other rows.

To determine indegree, add all the coordinates in a column and compare the result to the addition results from the other columns. The highest number wins in both cases.

Anyways, I hope this helps. Just remember I’m only human and can make a lot of simple mistakes.

 If I’m wrong please let me know, because this is the process I’m using in my program.