Part1:

Algorithm idea:

First generate a dictionary which key is edge and value is 1. Use triple nested loop, each loop goes through the entire graph. In the outer loop, we get our vertex of triangle, and next level we can get all the node connect to the vertex by checking if edge (vertex, node1) exist using the dictionary which generated before. And the most inner loop, we can get all node that are connect to the both vertex and node1 using the same strategy as second level which check edge(vertex, node2) and edge(node1,node2) exist. When the loop end we can get all the triangle.

Part2:

Algorithm idea:

Still can use triple nested loop, but this time, the outer loop is same which loop through all the node. But the second one, loop through each element of its adjacency list. For third loop go through each node in the same adjacency list that has a greater index number than the element we got from second loop. With second and third loop, we can get all the possible pair of two node that connect with the vertex. Then just check if the edge (node1, node2) exist. If yes we get a triangle. Therefore, for every vertex we can get all triangle connects to this specific points. As we go through all the node, so we get all the triangle.

Algorithm detail:

Assume we have the same dictionary of edge as in part1.

for vertex in Graph:

for (i = 0;i<len(Graph[vertex]);i++):

for (j=i+1;j<len(Graph[vertex]);j++):

try:

if edge[Graph[vertex][i],Graph[vertex][j]] ==1: 🡨check if edge exist

outputlist.append((vertex,Graph[vertex][i],Graph[vertex][j]))

except KeyErrorException:

continue

Runtime Analysis:

First for loop is clearly O(n), Second for loop the limit is length of the adjacency list which in worse case can be O(the longest edge number which defined in the question instruction as a triangle sign), and the third loop make pair with second loop. We can say that is still O(triangle) too. And the inner two nested loop can’t possibly larger than m which is the number of the total edge number, but can be.

So overall the runtime is going to be O(n\*min(m,trangle^2))

Proof idea:

I view all the node as a vertex of a triangle, and then check the all nodes which connect to it, making pair of two node at a time check if this two node are connected too. Therefore, I can at most actually go through all the edges for each vertex in order to make sure all the triangles had been found.