Name: Sumiya Arafin **Student ID: 20166010**

Program: M.Sc **Subject: Graph Theory**

Course Code: CSE708

Task-1:

1. At first initialize an empty stack and make a recursive DFS traversal for the all vertices of the given graph.
2. Then push the result of the DFS traversal to the stack.
3. Then compute the transpose graph.
4. Make a DFS traversal to the transpose graph in decreasing order of their finish time.
5. Then it output vertices as separate strongly connected components.

Task-2:

1. Declare all the vertices V of graph as unvisited. Create an empty data structure L*.*
2. Do visit(V) recursively to Visit all the vertex of the graph
   Here, If *V* is unvisited then:

   1. Declare *V* as visited.
   2. Do Visit(N) to visit the neighbor N of V.
   3. Add V to S.

   Else do nothing.

3. Assign (V, V) where (V, Parent) is recursively assigned for ordered elements V of S

   If *V* is not already assigned to a component:

   1. Assign *V* to the component as belonging and set its root as parent.
   2. Do assign (N, Parent) for all the neighbor N of V which are directed to the V.

   Else do nothing.

Task 3:

To show the correctness of the algorithm some relation described below:

P(a, b)= There is a path from a to b in the original path.

T(a, b)= There is a path from a to b in the transpose graph.

FA(a, b)= DFS finishes a after b'

If there is a path from a to b but not to other , DFS is going to finish after a. we can write this as P(b, A)^¬ P(a, b)= FA(b, a) taking the contrapositive we find:

¬ FA(b, a)= ¬(p/b,a)^¬P(a, b)

Getting rid some of the negative give us:

FA(a, b) = ¬P(b, a) ᵛ P(a, b)

Or in other words, if a comes before b in the order found in the algorithm, then either there is a path from a to b or there is no path between them in either direction.

Following algorithm, let's consider that while doing a DFS from a in the transpose graph we meet a node b(TP(a, b)). A must come before b in the order found earlier in the algorithm. Since otherwise we have already visited b and therefore ignore it while exploring a.

In such a case there are paths joining a to b in both directions and so they are part of same connected component.

We have shown that any two nodes identified as belonging to the same strongly connected component will be output as such to prove the correctness of the algorithm.

Each strongly connected component will have some node whose positions in the sort is less than that of every other node in the same strongly connected component. There will exist a path from that node to every other node in its strongly connected components. Additionally, at the beginning of the exploration of that first node, none of other node will have been explored yet. There two properties are exactly what guide DFS. So, it follows that DFS will find all of the nodes in that strongly connected component.