National University
of computer and emerging sciences

# Project Report

**Group Members:** Bazla Rashid 19I-1693

Oheed Imran 19I-0525

Syed Mustafa 19I-0544

**Submission Date:** **20-06-2021**

## Contents

# Task-01

## Number of nodes:

***Data Fetch():***
Extracted data from text file.

***DisplayTotalNode():***
Display total number of nodes from the text file.

## Number of edges:

Following function calculates the number of the edges.

***CalculateNumberofEdges():***
One for loop traverses over total number of nodes in graph and the other while loop through the adjacency list of every node.

## Source nodes:

Following function is used to calculate number of source nodes.

***DisplaySourceNodes():***
One for loop traverses the graph on the number of nodes and compares each node with the adjacency list of all other graph nodes inside second for loop.

## Sink nodes:

Following function is used, to calculate number of sink nodes.

***DisplayNumberOfSinkNodes():***
If adjacency list of a node is null then the outdegree of a node is zero and it will be a sink node.

## Isolated nodes:

Following function is used to calculate number of isolated nodes.

***DisplayIsolatedNodes():***
If the indegree and outdegree of node is zero then it will be an isolated node. For that there are two function that calculated the indegree and outdegree and return true or false. In DisplayIsolatedNodes() function it checks if both returns FALSE for a node then it is isolated node.

## Shortest path length distribution:

Following function is used, to calculate shortest path distribution.

***shortestpath (int startingnode):***
Initially all vertices of graph in visited array are said to be false and other array is

Dist[i]=-1;

At the start dist[source] is initialize with zero as distance from a vertex to itself is zero and marked as visited. BSF is used to traverse the connected components of a graph and marked as visited, dist array is updated when it is not visited, and it is updated with the value with the parent vertex in dist array adding one in it. This Function is return dist array and it is called in function findingshortestpath() where we check if a node is not updated from -1 then it called again to check the shortest path or other connected components as well.

## Diameter of graph:

Following function is used,

### *DiameterofTheGraph():*

It contains an array visited of size total number of nodes and shortestpath() function is called in it which returns Dist array which is store in visited array. We traverse the array to find the maximum distance in each connected component, this process is repeated for all the components in a graph.

# Task-02

## IndegreeDistribution():

One for loop traverses the graph on the number of nodes and the other loop is to traverse the adjacency list of nodes and calculating the out degree of every node.

This is then inserted into the distribution class (`class NLDClass)` and stored then through display function distribution is displayed.

## OutDegreeDistribution():

One for loop traverses over the number of the nodes in the graph and the other for loop checks whether the adjacency list of that is empty or not if it is empty it increments the out degree and insert in distribution class (`class NLDClass).`

# Task- 03:

## Strongly connected components:

Following are the functions used to calculate the number of strongly connected components.

- `bool BFSForInalgorithmSCC(T startnode, T value_temp)`
- `adjacencyMatrix<T>* InalgorithmForScc()`
- `AdjencencyList<T> EnhanceBfsForoutdegreeCalling(T startnode)`
- `adjacencyMatrix<T>* outalgorithmForScc()`

### StronglyconnectedAlgorithm():

This function calls the above listed functions. InalgorithmForScc() finds different nodes from which source vertex can be visited and stored them in link list. outalgorithmForScc() finds all those nodes that can be access be accessible from the start vertex.

Then we find the intersection of the link lists returned by InalgorithmForScc() and outalgorithmForScc(). We then find the unique elements and then count the largest number of nodes link list and return its size. This will be the largest strongly connected component.

## Weakly connected components:

For weakly connected we will take graph as an undirected graph.

### *weaklyConnectedComponents():*

This function finds the weakly connected components, one loop traverses over the total number of nodes in the graph and AdjencencyList<T> EnhanceBfsForoutdegreeCalling(T startnode) is called that returns all those accessible nodes to the startnode and then they are saved in another adjacency list. Later unique elements are found and the size is inserted into NLDClass for size distribution.