

**CS218-Data Structures**

**Project Report**

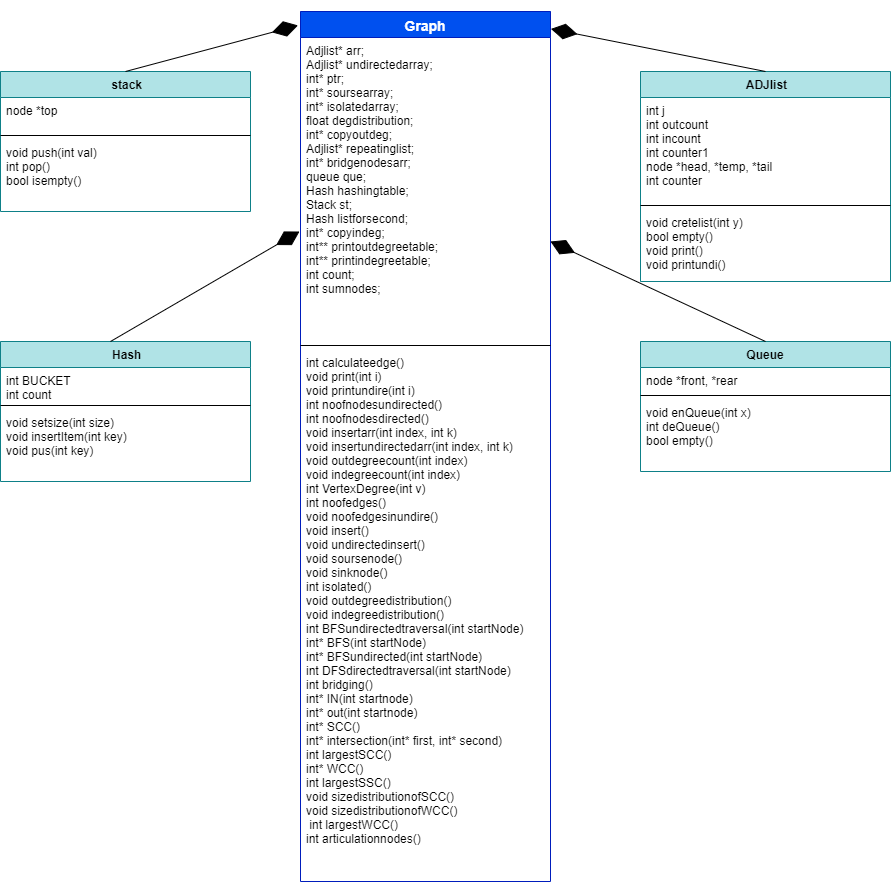
**Submitted By:**

Hanzla Sibghat 19I-0515(A)

Muhammad Umer 19I-0521(C)

Waqas Ahmad 19I-0606(A)

**UML Diagram:**



**About our project:**

In this project we have learned about the different algorithms applied to traverse a graph like BFS (Breadth First Searching) and DFS (Depth First Searching) and different properties of the graph. We divided the structure of the program into 5 major classes which are ADJ list, Queue, Stack, Hashing and Graph.

Basically, in the smallest data set the graph is in the form of directed graph and we have to store it in the list. And for some function its used as the directed graph e.g. for taking source node, sink node, isolated nodes, indegree and outdegree distribution, SCC and its distribution. And for other functions like bridge edges, articulation nodes and specially for WCC we have used the graph as Undirected.

The real-life example of this project scenario is **Google maps**. Say that the **nodes** are different shops located in a city and the **edges** between them are the roads/routes that connects them. If a road/route between two shops is blocked then it is an example of a **bridge**. The example of **sink** is a road that has a closed-end/Dead-end. The example of an **isolated node** is like an island that is disconnected from the city.

The smallest data set that we worked on is a disconnected graph.

**Queue:**

The main rule of the queue is FIFO (First in First Out) and the **time complexity** for **queues** is O (n).

**Stack:** The main rule of the stack is LIFO (Last in First Out) and the **time complexity** of stacks is O (1). This technique is best in insertion and deletion of nodes.

**ADJ List:**

We have used the structural framework of Adjacency List instead of Adjacency Matrix because adjacency Matrix has the time complexity of O(|V^2|) whereas the adjacency List has time complexity of O (|V| + E). The adjacency Matrix is a contiguous memory location in which most of the memory space is left unused and hence is a waste of the memory. Whereas the adjacency List is a dynamic memory, an array which has the addresses of lists stored at each index and the memory is allocated for the data that is to be stored in front of each index as required so, none of the allocated memory space is left unused.

In our project the adjacency list is used to store the node at a unique index and its neighboring nodes against its list.

**Hash:**

We used hashing instead of linked (list which traverses through the whole array) in some functions to insert values in array to traverse to the exact index and minimize the time complexity.

**Graph:**

1. **No of nodes**

In this function we have calculated the total number of nodes given in the data set. In smallest data set we have total number of **5242** nodes.

1. **Node Edges**

In this function we have traversed all the nodes and their respective neighbors to calculate the number of edges. In the smallest data set of **5242** nodes, we have **28980** edges.

1. **Source Node**

In this function we have checked that any node that has zero indegree is Source Node. It takes some time to calculate because we traverse each node to calculate the source. In the smallest data set of 5242 nodes, we have **zero source node.**

1. **Sink Node**

In this function we have checked that any Node have some head or not. If a Node have a head (NULL) or have zero outdegree then it is sink node. And it takes n times traversal to search the node which have head NULL. In the smallest data set of 5242 nodes, we have **zero sink node.**

1. **Isolated Node**

In this function we have to check if the node have zero indegree and zero outdegree. If a node has both the zero outdegree and indegree then it is isolated node. In the smallest data set of 5242 nodes, we have **zero Isolated node**.

1. **Bridge Edge**

To check the bridge, remove edge one by one then take BFS traversal and compare it with original BFS of graph if

The answer of original BFS is greater than the BFS of removed edge graph then it is bridge edge.

It is time taking process because it takes the BFS of all nodes one by one and takes too much time around **1.5 hours**. It has many bridge edges it takes time to calculate I have only waited 40 mins and its only have checked 2000 nodes only.

1. **Articulation nodes**

To check the articulation node, we remove the vertex one by one and then compare it with the original BFS or DFS if the answer

is less then equal to the original BFS - 2 then it is articulated node.

It takes around 20 mins on smallest data set of 5242 nodes. It has around 1100 to 1200 articulation nodes.

1. **Shortest path distribution:**

In this function we start from a unique node every time and find the number of node traversals required to traverse through the whole graph. In this function the traversal method we used is BFS (Breadth First Traversal).

The function takes 1 minute to traverse 300 nodes and approximately 18 minutes to traverse through 5242 nodes.

1. **Diameter**

In this function, we find path length of each node and select the node which has the maximum path length is the diameter of graph. This function also takes 1 minute to traverse 300 nodes and approximately 18 minutes to traverse through 5242 nodes.

1. **Indegree distribution**

This function is used to check how many of nodes have e.g., 1 indegree and how many have 5 degrees etc.

The formula used in calculating the indegree distribution is

**V(k)/V**, for example in our data set total number of nodes are 10 and V should be 10 and V(k) is the specific node and after searching it indegree, divide it by V.

1. **Outdegree distribution**

This function is used to check how many of nodes have e.g., 1 outdegree and how many have 5 outdegree etc.

The formula used in calculating the outdegree distribution is

**V(k)/V**, for example in our data set total number of nodes are 10 and V should be 10 and V(k) is the specific node and after searching it outdegree, divide it by V.

1. **Display size of largest SCC**

By taking BFS from all nodes one by one and checking if the source node is present in the BFS we saved that node in the array after all iteration take intersection with the source node BFS and that is the SSC component and the largest in that array size is the largest set.

This is time taking process as each BFS takes less than 1 sec in smallest data set so the dataset of 5000 nodes can take 40 to 60 minutes to complete SCC.

1. **Display size distribution of SCC**

In this we have to take all the nodes size of traversing in the form of table after taking probability.

1. **Display size of largest WCC: (Undirected Graph)**

For WCC we consider the graph as Undirected and after this we take BFS of every node and check if a node visits all the connected edges and the largest to connected to their neighbor is Largest WCC. It takes around 20 to 24 mins for the smallest dataset. And largest WCC is 4158

1. **Display size distribution of WCC: (Undirected Graph)**

In this we have to take all the nodes size of traversing and display in the form of table after taking probability.