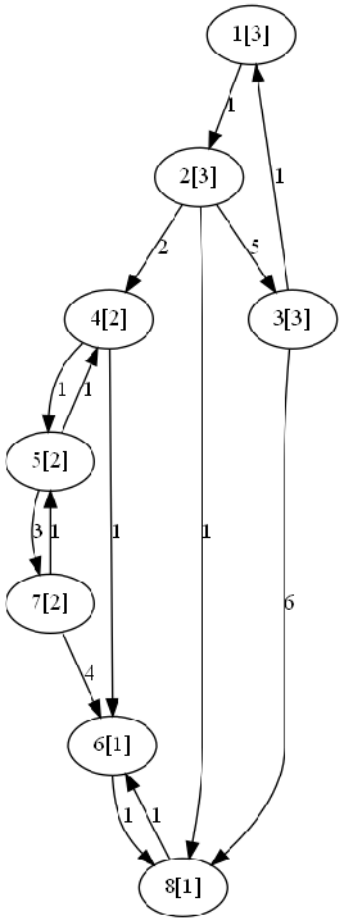
DS LAB ASSIGNMENT 4

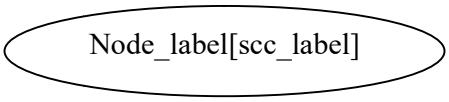
GRAPH

214101022

Graph representation:

- Graph node labels start from 1 to n. Where n is the number of nodes/vertices.
- **Node structure for every part of assignment is different.**
- Example of strongly connected component(scc) graph is given below.



Node structure here is 

We can see the strongly connected component in the graph.

(1,2,3) is one component (4,5,7) is one component and (6,8) is one component.

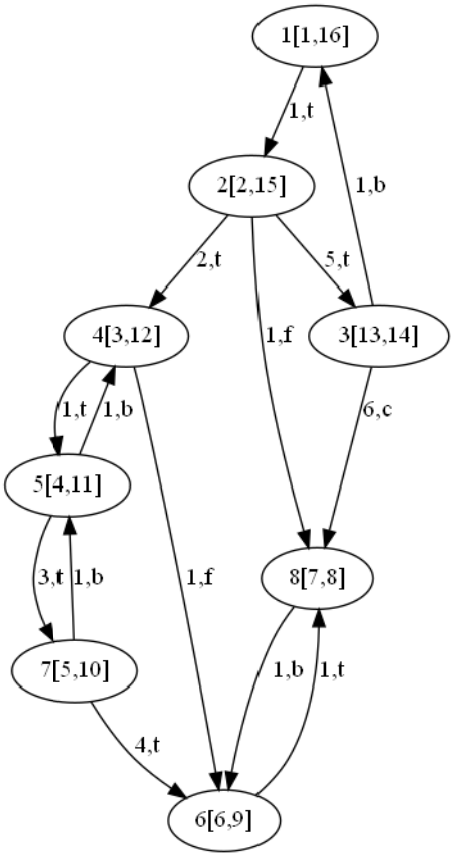
PART 1 : Perform DFS traversal on the graph and classify all the edges.

Node structure and edge structure is different here. Perform dfs and we will get the graph. Below is the example of node structure and edge structure.

Node structure:- Node_label[start_time,end_time]

Edge :- (edge weight , edge type)

Example of graph for part 1,

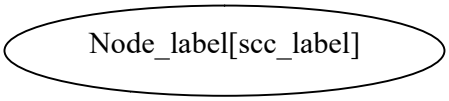


- t = tree edge (for ‘u’ to ‘v’ edge if v is not visited in dfs yet then it is tree edge)
- f = forward edge (for ‘u’ to ‘v’ edge if v is already visited in dfs and currently not present in stack and start time of v is greater than u)
- b = backward edge (for ‘u’ to ‘v’ edge if v is already visited in dfs and currently present in stack)
- c = cross edge (for ‘u’ to ‘v’ edge if v is already visited in dfs and currently not present in stack and start time of u is greater than v)

Depending on traversal we may get different edge types on same graph.

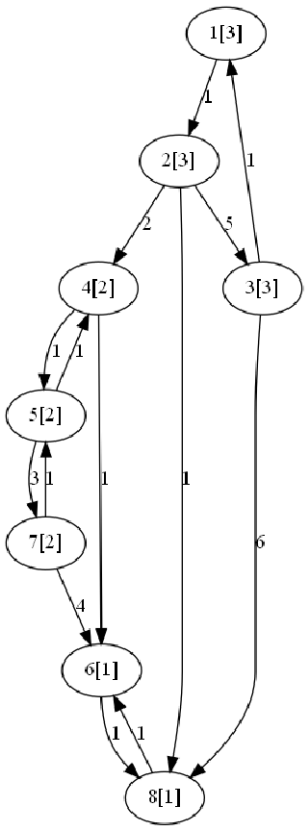
TIME COMPLEXITY : O(V + E) (DFS traversal)

PART 2 : Find all strongly connected components using Tarjan's algorithm.

Node structure :- 

Edge :- edge weight

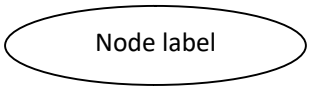
A directed graph is strongly connected if there is a path between all pairs of vertices. A strongly connected component (scc) of a directed graph is a maximal strongly connected sub-graph. For example, there are 3 scc's in the following graph.



TIME COMPLEXITY : O(V + E) (DFS traversal)

- Low value is calculated in dfs and nodes are pushed back in stack.
- Low(v) value is minimum of:
- (1)disc[v], (2) the lowest disc[u] among all back edges(v,u), (3) the lowest low[u] among all tree edges(v,u),
- If for a vertex v, the low value is equal to its discovery time then that vertex is part of a strongly connected component. The vertices are then popped out from the stack until the popped element is v and all these vertices from a strongly connected component.

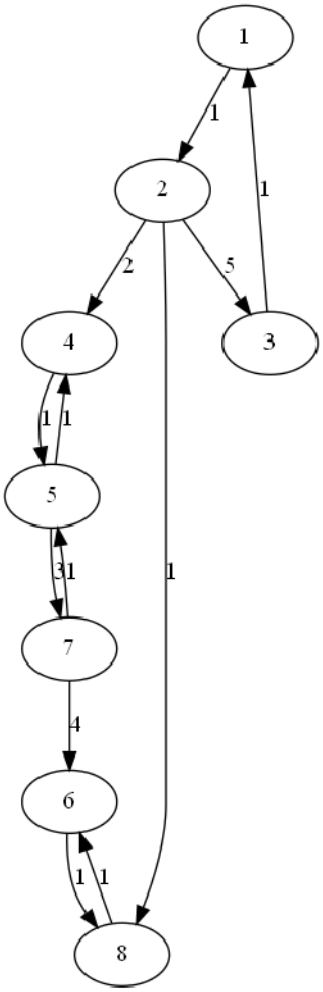
PART 3 : Given a directed graph G = (V, E), create another graph G'= (V, E'), where E' is a subset of E, such that (a) G' has the same strongly connected components as G, (b) G' has the same component graph as G, and (c) E' is as small as possible.

Node structure :- 

Edge :- edge weight

- Find strongly connected component using Tarjan's algorithm.
- Create a matrix of nxn (n is the number of components).
- Keep track of edges from one component to another.
- Only keep one edge from one component to another and remove all others.
- Keep two more arrays of size n to keep track of column and row of matrix. If all values of arrays are 1 in two arrays at the end then graph is semi-connected as all the components are somehow connected.

TIME COMPLEXITY : O(V + E) (DFS traversal)



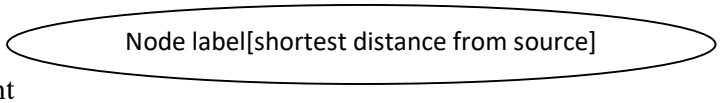
PART 4 : A directed graph G = (V, E) is semi-connected if, for all pairs of vertices u, v in V, we have either a path u ---> v or a path v ---> u. Design an efficient algorithm to determine whether or not G is semi-connected. Implement your algorithm and analyze its running time.

- Find all strongly connected components using tarjans algorithm.
- Either we can create a scc graph where one scc is considered as one vertex.
- Or we can consider original graph where one scc is considered one vertex.
- Find the indegree of every vertex or component using graph traversal.
- Indegree of a component is the edges coming from other component in the graph.
- Check if there is a component where indegree is 0.
- Traverse graph once again using dfs.
- If all the components can be visited means graph is semi-connected.
- If one or more of the components can not be visited then graph is not semi-connected.

Output of the part 4 is YES graph is semi-connected or NO graph is not semi-connected.

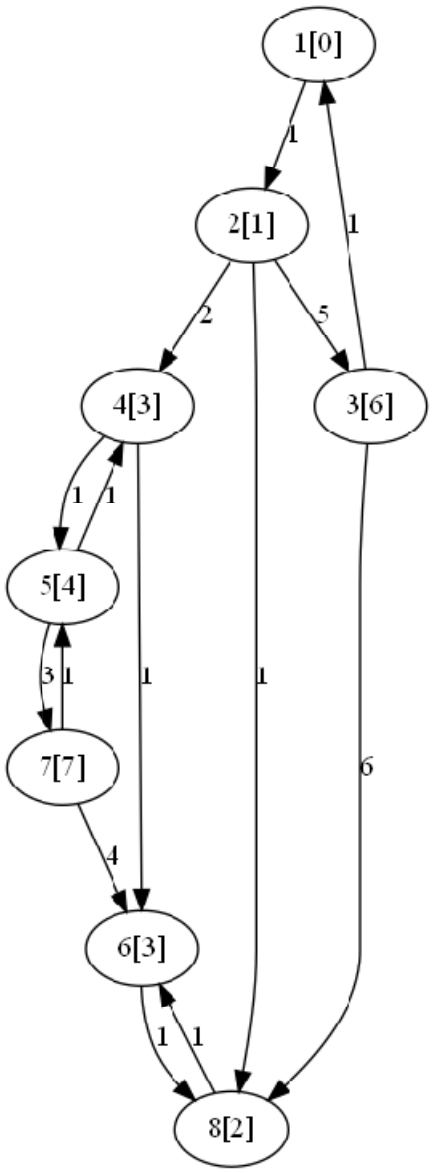
TIME COMPLEXITY : O(V + E) (multiple DFS traversal)

PART 5 : Dijkstra's algorithm

Node structure :- 

Edge :- distance/weight
Initially all distances are 1,000,000.

- Take an input from the user.
- Input is the start node or source node labelled from 1 to n. (n is the number of vertices)
- Form the source we will apply dijkstra.
- Create a heap which will keep track of all the shortest distances present from current node to other.
- Create a set which will keep track of all shortest distances from source.
- While heap doesn't become empty
- a) Extract minimum distance vertex from heap. Let the extracted vertex be u.
- b) Loop through all adjacent of u and do following for every vertex v
- If dist[v] > dist[u] + weight(u, v)
  (i) Update distance of v, i.e. dist[v] = dist[u] + weight(u, v)
  (ii) Insert v into the heap (Even if v is already there)



The time complexity remains O(ElogV)) as there will be at most O(E) vertices in heap and O(log E) is same as O(log V)

TIME COMPLEXITY:- O(E logV)