**ASSIGNMENT 4 GRAPHS**

**QUESTION 1: Perform DFS traversal on the graph and classify all the edges. After DFS show the annotated graph where each vertex is labeled with the DFS start and end time and each edge is labeled with its edge type (tree/forward/back/cross)**

**ALGORITHM USED:**

```
DFS(G)
for each vertex u ∈ G.V
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V
    if u.color == WHITE
        DFS-VISIT(G, u)

DFS-VISIT(G, u)
time = time + 1
u.d = time
u.color = GRAY
for each v ∈ G.Adj[u]
    if v.color == WHITE
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK
time = time + 1
u.f = time
```

---

**input used--**

**10 17 ← vertices edges**

**0 1 2 ← edgeU edgeV weight**

**0 3 2**

**1 2 2**

**1 4 2**

**2 0 2**

**2 6 2**

**3 2 2**

**4 5 2**

**4 6 2**

**5 6 2**

**5 7 2**

**5 8 2**

**5 9 2**

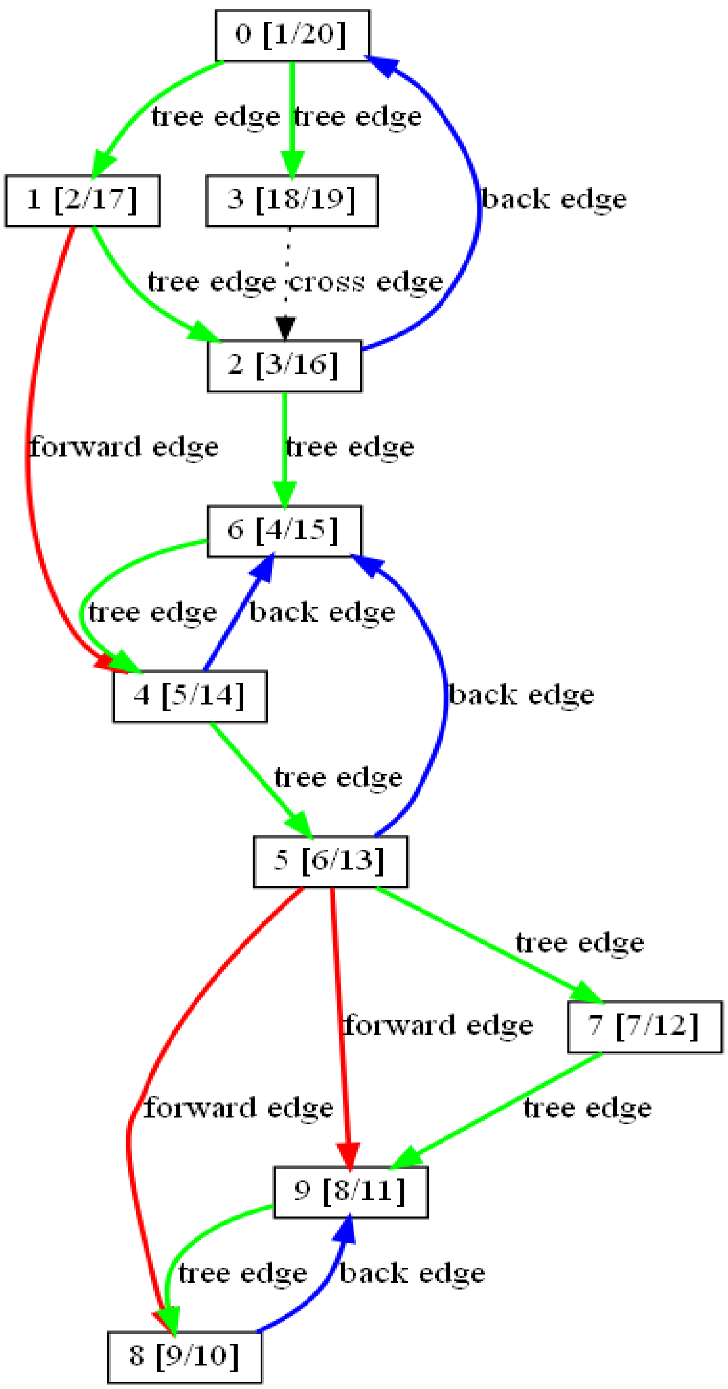
**6 4 2**

**7 9 2**

**8 9 2**

**9 8 2**

**Output:**

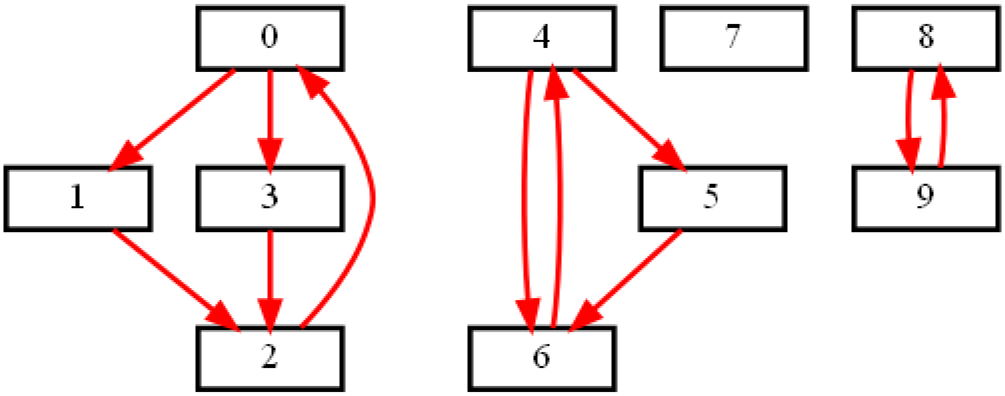


**NODE STRUCTURE: IN DFS TRAVERSAL**

| NODE NO. [START TIME / END TIME ] |
|---|

**QUESTION 2: Find all strongly connected components using Tarjan's algorithm**

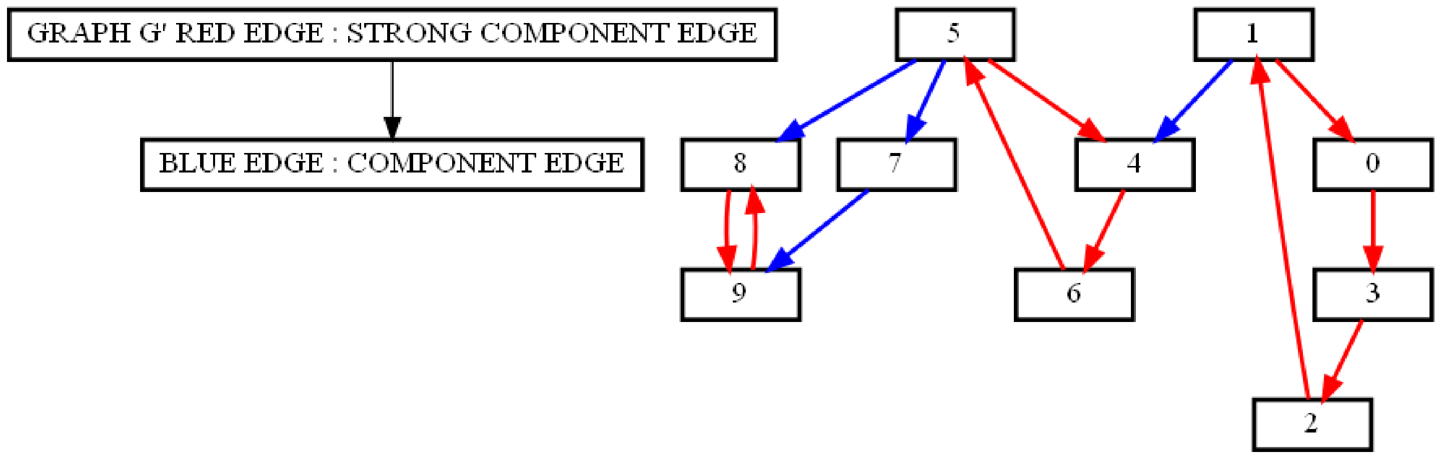
**Output:**




**QUESTION 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.**

**Solution:**

- **Let C1, C2, · · · , Ck(k ≥ 1) be the strongly connected components of G. Let Cl = {v1, v2, · · · , vi}. Now in G', add one edge between {vjmodi, vj+1modi}, j = 1, 2, · · · i from vjmodi to vj+1modi. This creates a directed cycle between the i vertices in the strongly connected component Cl.**
- **In addition to the above, add one each from some vertex u in a strongly connected component Ci to a vertex v in another strongly connected component Cj in the direction from u to v iff any vertex in Ci is connected to a vertex in Cj.**
- **So condition (a) of the question is met here as all the vertices in the directed cycle form a strongly connected component.**
- **Condition (b) is easily met by the construction of the component graph.**
- **For (c), directed cycle is a strongly connected graph with least number of edges. So each of the strongly connected components in G' have the least number of edges between vertices in the same component.**
- **For meeting condition (b), there must be at least one edge between strongly connected components that are connected and in our construction we add only one edge.**
- **To construct G', first obtain the strongly connected components of G and add the edges as specfied in the above. The algorithm takes O(V +E) time as the time to obtain strongly connected components in O(V + E) and for the construction we need only O(V +E) time again.**
- **Output**




•

---

**QUESTION 4: A directed graph G = (V, E) is semiconnected 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 semiconnected. Implement your algorithm and analyze its running time**

**Solution**

**1. Run STRONG-CONNECTED-COMPONENTS(G) (USING TARJAN's algo in question 2).**

**2. Take each strong connected component as a virtual vertex and create a new virtual graph G' which becomes a DAG o f graph G'.**

**3. Run TOPOLOGICAL-SORT(DAG-G').**

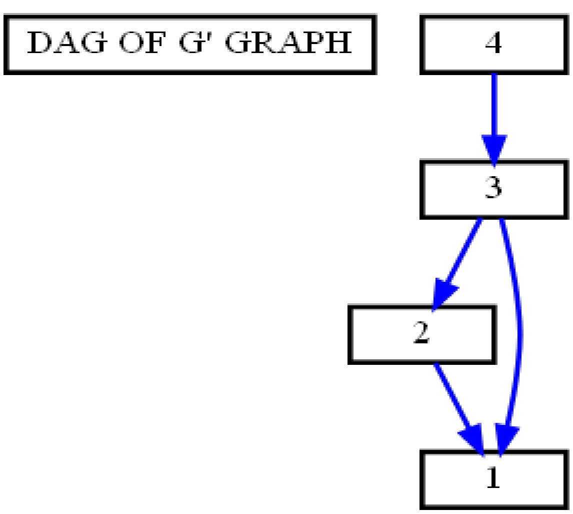
**4. Check if for any consevutive vertices (Vi, Vi+1) in a topological sort of G', there is an ede (Vi, Vi+1) in graph G'. if so, the original graph is semiconnected. Otherwise, it isn't.**

**ANALYZE:**

**Consider consecutive vertices Vi and Vi+1 in G'. If there is no edge from Vi to Vi+1, we also conclude that there is no path from Vi+1 to Vi since Vi finished after Vi+1. From the definition of G', there is no path from any vertices in G who is represented as Vi in G' to those represented as Vi+1. Thus G is not semi-connected. If there is an edcge between any consecutive vertices, there is an edge between any two vertices. Therefore, G is semi-connected.**

**RUNNING-TIME: T = T(SCC) + T(TOPOLOGICAL-SORT) = O(E +V).**

**Output:**




---

**IT IS SEMICONNECTED**

**Topological sort 4 3 2 1**

**QUESTION 5: Implement Dijkstra's shortest path algorithm (in O(Elog V) using a priority queue) to find a shortestcost path between a source vertex s to any target vertex t.**

**Output:**

**NODE STRUCTURE**

| NODE NO. [MINIMUM DISTANCE] [NODE FROM WHICH IT CAME FROM] |
|---|