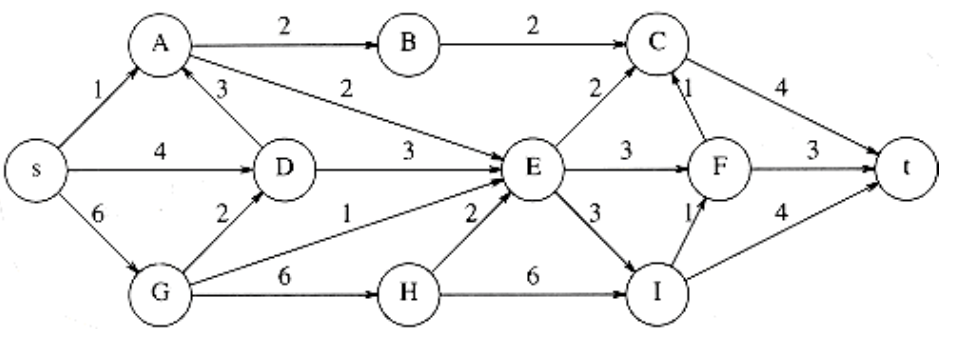
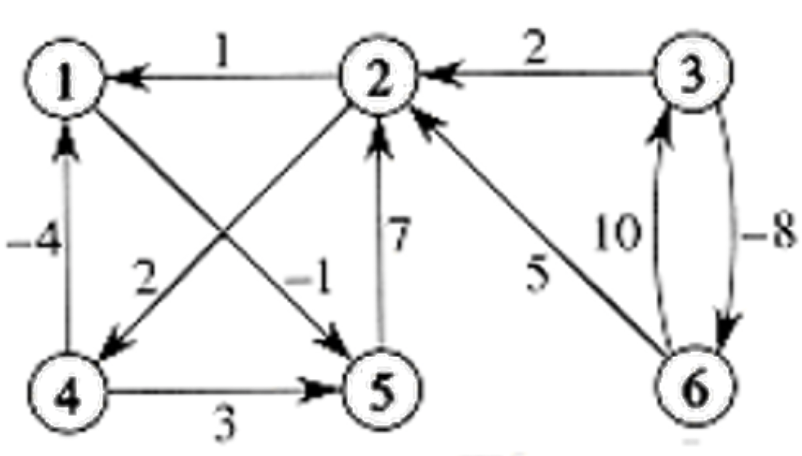
1. Find a topological order for the graph in the following figure   
   **S-->G-->D-->A-->B-->H-->E-->I-->F-->C-->t**  
   

The topological order is as such because **S** has no receiving inputs, therefore we can remove it. The next node we can remove is **G** as **S** is the only connector. Once **S and G**  are removed as projecting nodes, **D** no longer has any inputs, therefore we can remove it. Afterwards, **A** will have no inputs because **S & D** are removed, therefore we can remove it (by this point we are at **S-->G-->D-->A**). Continuing with **A** and its removal, we can remove **B** because we have no incoming connectors (**S-->G-->D-->A-->B**). Now we have to turn our attention to the next node without incoming receptors, **H,** and we can remove it ((**S-->G-->D-->A-->B**→**H**), and now we can remove **E,** as all of its connectors are removed. Now that **E & H** are no longer projecting onto **I**, we can remove it as well (**S-->G-->D-->A-->B-->H-->E-->I→).** With **I & E** eliminated, we can remove **F (**(**S-->G-->D-->A-->B-->H-->E-->I→F)**. And, now that **B & F** are removed and no longer connecting onto **C**, we can remove it, and this will allow us to complete the topological order, with **t** being the last node without any connectors, and resulting in this topological order: **S-->G-->D-->A-->B-->H-->E-->I-->F-->C-->t**

1. Find all pairs of shortest distances for the following graph

****

**D0 (Initial pairs) D6 (FINAL STATE/PAIRS)**

0 ∞ ∞ ∞ -1 ∞ 0 6 ∞ 8 -1 ∞

1 0 ∞ 2 ∞ ∞ -2 0 ∞ 2 -3 ∞

∞ 2 0 ∞ ∞ -8 5 -3 0 -1 -6 -8

-4 ∞ ∞ 0 3 ∞ -4 2 ∞ 0 -5 ∞

∞ 7 ∞ ∞ 0 ∞ 5 7 ∞ 9 0 ∞

∞ 5 10 ∞ ∞ 0 3 5 10 7 2 0

This answer was achieved through this pseudocode:

// BEGIN Floyd-Warshall algorithm for finding shortest paths

FOR ALL v IN {1, 2, 3, 4, 5, 6}

// Set distance from each node to itself to 0

d[v][v] ← 0

// Set initial distances based on direct edges between nodes

FOR ALL edges (u,v) with weight w in the graph

d[u][v] ← w

// Check paths through each node as an intermediate

for k from 1 to 6

// For all pairs of nodes (i, j)

for i from 1 to 6

for j from 1 to 6

// If a shorter path is found via node k, update distance

if d[i][j] > d[i][k] + d[k][j]

d[i][j] ← d[i][k] + d[k][j]

// End of path comparison

end if

// END of algorithm, d contains shortest paths between all pairs

// Example run for updating matrix D1 using node 1 as an intermediate node

FOR i from 1 to 6

FOR j from 1 to 6

// Update d[i][j] if a shorter path through node 1 is found

d[i][j] = min(d[i][j], d[i][1] + d[1][j])

// The resulting D1 matrix after considering paths through node 1:

0 ∞ ∞ ∞ -1 ∞

1 0 ∞ 2 0 ∞

∞ 2 0 ∞ ∞ -8

-4 ∞ ∞ 0 -5 ∞

∞ 7 ∞ ∞ 0 ∞

∞ 5 10 ∞ ∞ 0