Manish Semwal

204101032

Ans 1:

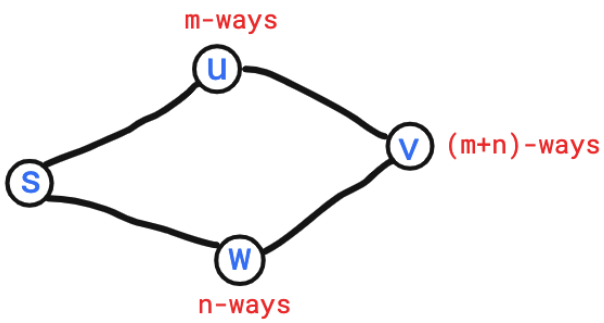
a) num[v] stores the number of shortest paths from the source vertex 's' (supplied as the input of the algorithm) to all the vertices in G.

Justification: The algorithm is a modified BFS, where we're maintaining list 'num' which stores a value for each vertex.

For a graph with 1 vertex, 'num' gives the no. of path from it to itself. Which is true since we're initializing num for source to 1.

If the nodes at the other end of the edge i.e. if 'u' is dequeued then for every edge (u, v), if node 'v' is unexplored yet then we simply do num[v]=1, which means only 1 shortest path is known yet for 'v'

Otherwise, let the algorithm has run to a stage that it has calculated 'num' for two nodes 'u' and 'w' and no. of shortest paths from 's' (source node) to 'u' and 'w' are num[u] and num[w] resp.



Now, for node 'v' the no. of shortest paths form 's' are num[u]+num[w].

Similarly, it can be extended to any no. of nodes as predecessor of 'v'.

b) Time Complexity is O(V + E).

where V is number of nodes in G and E is the number of edges in G.

This is because at every node we're checking all its neighbors which sums up to '2E' and for this we are visiting every node exactly once which requires 'V' time. Therefore, overall time complexity is V + 2E = O(V + E).

Ans 2:

a) 'IsGood' holds true if that nodes is successor of its predecessor in Dijktrsa's single source shortest path algo, otherwise isGood is false for that node.

b) Time Complexity is O(E*logV).

where V is number of nodes in G and E is the number of edges in G.

This is because in the algorithm is modified Dijkstra's Single Source Shortest Path algorithm.

Here, we're performing V delete operation. And for each delete operation we're visiting its neighbors which in total is 2E. For each edge we're doing one decrease key (logV time) operation in worst case. This requires, O(E*logV) time.