1. 1. To prove that if G contains a negatively weighted cycle,then there exists $u \in V$ such that dist(u) is updated at the $|V|$-th iteration, we can just prove that if there doesn't exist $u \in V$ such that dist(u) is updated at the $|V|$-th iteration, then G doesn't contain any negatively weighted cycle:

      If there's no $u \in V$ such that dist(u) is updated at the $|V|$-th iteration, then there's no $|V|$-edge-path, which means all shortest paths are no longer then $|V|$-1 edges. [1]

      Assume that there is a negative cycle, then there will be shortest paths for vertices in the cycle that have infinite edges, which disagrees with [1].

      So, if there doesn't exist $u \in V$ such that dist(u) is updated at the $|V|$-th iteration, then G doesn't contain any negatively weighted cycle, which means if G contains a negatively weighted cycle,then there exists $u \in V$ such that dist(u) is updated at the $|V|$-th iteration.

   2. V={s,a,t}

      E={(s,a,1),(a,t,-1),(t,a,-1)}

      initial:{0,inf,inf}

      round1:{0,1,inf}

      round2:{0,1,0}

      round3:{0,0,0} dist(t) is not updated

   3. Adapted Bellman-Ford algorithm:

      1. Initialize dist(s) = 0, dist(u) = ∞ for each u ≠ s, and a boolean variable has_negative_cycle = False.
      2. For i = 1 to $|V|$ - 1: a. For each edge (u, v) ∈ E: i. If dist(u) + w(u, v) < dist(v), set dist(v) = dist(u) + w(u, v)
      3. For each edge (u, v) ∈ E: a. If dist(u) + w(u, v) < dist(v): i. Set has_negative_cycle = True
      4. If has_negative_cycle: a. Run the Bellman-Ford algorithm again with the updated dist values b. If dist(t) is updated in this second run, return True (there is an s-t path containing a negatively weighted cycle) c. Else, return False (there is no s-t path containing a negatively weighted cycle)
      5. Else, return False (there is no negatively weighted cycle)

      The first Bellman-Ford detects if the graph has a negatively weighted cycle. The second detects if there is a path from s to t that goes through the negatively weighted cycle by checking if dist(t) is updated.

      The adapted Bellman-Ford algorithm consists of two runs of the original Bellman-Ford algorithm, so its time complexity of the adapted algorithm is also $O(|V| \cdot |E|)$.

2. Algorithm:

   1. Loop:

1. Perform Kosaraju algorithm
    1. Perform a depth-first search (DFS) on the graph G, starting from any vertex. During the DFS, mark each vertex as visited when it is finished being explored.
    2. Create a new graph G', where the edges of G are reversed. That is, for each edge (u, v) in G, add the edge (v, u) to G'.
    3. Perform a DFS on G', starting from the last vertex visited in step 1 that has not yet been explored in G'. Each tree in the DFS forest of G' corresponds to a strongly connected component of G.
2. Merge the SCCs into vertices, mark them as s or t if the SCC contains s or t.
3. if no SCC is found, end loop

2. Modified DFS, which skips t if it is not the last vertex.
    1. s and t may be the same vertex now. Perform a DFS starting from s, and at each step, mark the current vertex and edge as visited.
    2. If t is reached, check if all vertices in V have been visited. If yes, return true. Otherwise, continue the search from any visited edge or vertex that has not been fully explored.
    3. If all edges and vertices reachable from s have been explored without reaching t or visiting every vertex, return false.

Proof of correctness:

We can merge SCCs because vertices and edges can be visited more than once. After merging SCCs, it will be impossible for us to visit a vertex for more than once, so we can do the visited marking process, and use the modified DFS to determine the answer. If we can walk f

Time complexity:

For the Kosaraju algorithm, the time complexity is O(|V|+|E|). For the modified DFS, the worst time complexity is also O(|V|+|E|), because for merged V' and E' , |V'|<=|V|, |E'|=|E|.

So the total time complexity is O(|V|+|E|).

3. 1. If undirected G is a tree, both DFS and BFS trees are (V,E), G is a good graph.

       If undirected G is a good graph, DFS and BFS tree of G are the same. Assume that the graph is not a tree, pick a cycle C in the graph. At least two vertices in the cycle are or the same distance to s, which means they are in the same tier in BFS. and there's no path that follows the BFS tree downwards from one of the two vertices to another. However, in DFS tree we always go downwards from one of the two vertices to another, which is counter to the former claim, so the graph is proved to be a tree.

    2. Disproof:

       s--->a--->c--->b

       s--->b

Both BFS tree and DFS tree are s--->a--->c s--->b

G is a good graph.

dist(c)>dist(b) is counter but c is before b topologically.

3. Disproof:

   s--->a--->b--->d

   s--->b

   a--->c--->d

   L: s a b c d is both ascending order of distance to s and topological order.

   BFS:

   s--->a--->c

   s--->b--->d

   DFS:

   s--->a--->c--->d

   a--->b

   or

   s--->a--->b--->d

   a--->c

   G is not a good graph.

4. 1. Algorithm:

      Start a BFS from s, mark all neighbors of s as 1,2,3,...,k, also, mark the vertices first found by BFS starting from a vertex marked i as i. Edges that have been gone through are marked as visited so that they won't be used again. If a BFS from a vertex meets a vertex that has been marked differently for the first time, then we have found the smallest cycle. If no such thing happen when all the vertices are visited, then the graph is a tree and does not contain a cycle.

      Proof of correctness:

      The cycle found in the BFS process must contain s, as the two ways of BFS emerge from different neighbors of s. Cycles that do not contain s won't be presented as the result because it is demanded that the vertices are marked differently. The smallest cycle's edges won't be marked before found, so there is no smaller cycle.

      Time complexity:

      The algorithm has the same time complexity as BFS, which is O(|V|+|E|).

   2. Algorithm:

Initialize all vertices' weight as inf and s as 0, initialize the smallest cycle's weight as inf. Also start a BFS from s, mark all neighbors of s as 1,2,3,...,k, and also mark the weight of the edges on the vertices, mark the vertices first found by BFS starting from a vertex marked i as i, also mark the vertices weight as the sum of the weight of the edge passed through and the sum of the previous weight.

If a BFS from a vertex meets a vertex that has been marked differently, check the sum of two weights of the same vertices, compare it with the weight of current smallest cycle, update if smaller, at the same time compare the original weight of the vertex and the new weight, update of the new one is smaller. If a BFS from a vertex meets a vertex that has been marked the same, compare the two weights and take the smaller one.

Proof of correctness:

Similar to 1, the cycle found in the BFS process must contain s, as the two ways of BFS emerge from different neighbors of s. The vertex mark and weight denotes which of neighbor of s has the branch of BFS emerged and the total weight of the route from s to it. The updating ensures that the vertex is marked with the currently smallest weight, so that when a new BFS branch from a different neighbor comes, the weights can be added to measure the currently smallest cycle that passes through two neighbors of s, s, and the vertex. The update of the global smallest cycle ensures that the smallest cycle is found.

Time complexity:

The worst condition is in a fully connected graph. There will be |V|-1 branches from s, each branch continues to compare and check the sum for |V|-2 times. So the time complexity is $O(|V|^2)$.