**Question 2:**

Yes, Dijkstra's algorithm is proficient in discerning the shortest paths from an initial vertex to all subsequent vertices within a graph, under the specialized condition wherein the sole occurrences of negative edges are those emanating from the initial vertex.

**Justification:**

1. **Non-negative edges after the start:**

Since the negative edges only emanate from the starting node ss, there are no negative edges in the remainder of the graph. This ensures that once we move away from the start, the typical assumptions of Dijkstra's algorithm (non-negative weights) hold, allowing the greedy selection of vertices based on the currently known shortest paths to work correctly.

1. **Greedy selection remains valid:**

The algorithm always picks the vertex with the minimum tentative distance. Since negative weights are confined to edges leaving ss, it won’t disrupt the greedy choice's validity as the algorithm progresses. No further negative edges will suddenly reduce the cost of reaching a node that’s already been visited.

1. **Correct distance propagation:**

The distances are propagated correctly even when there are negative edges at the beginning. When each node is investigated according to the increasing order of its tentative distances, the lack of negative edges in the remaining portion guarantees that each node's shortest distance is determined correctly and doesn't require readjustment.

1. **Absence of negative cycles:**

Since negative edges are restricted to leaving the starting node and no other negative edges exist in the graph, there are no negative cycles. Therefore, the condition that usually necessitates the use of Bellman-Ford algorithm doesn’t apply here.

***Conclusion:***

Given that the only negative edges are those leaving the starting node s and the rest of the graph contains non-negative edges, Dijkstra's algorithm can still operate correctly. The fundamental operations of the algorithm, such as greedy selection and distance propagation, remain valid, ensuring the correct determination of the shortest paths from s to all other nodes in the graph.

**Question 3:**  
The Floyd-Warshall algorithm is a dynamic programming algorithm that finds the shortest paths between all pairs of vertices in a weighted graph. It has a time complexity of O(n^3).

The algorithm works by iteratively updating a distance matrix, dist, which contains the shortest distances between all pairs of vertices. At each iteration, the algorithm checks if the shortest path between two vertices can be improved by going through a third vertex.

The algorithm terminates when the distance matrix no longer changes. At this point, the distance matrix contains the shortest distances between all pairs of vertices.

The following is a brief explanation of the algorithm:

1. Initialize the distance matrix, dist, such that the distance between each vertex and itself is 0 and the distance between each vertex and every other vertex is infinity.
2. for k in range(n): for i in range(n): for j in range(n): dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j])
3. return dist

The Floyd-Warshall algorithm can be used to find the shortest cycle in a directed graph by checking if the distance between any vertex and itself is less than infinity. If it is, then there is a cycle that starts and ends at that vertex. The length of the shortest cycle is equal to the distance between the vertex and itself.

The following is a brief explanation of how to use the Floyd-Warshall algorithm to find the shortest cycle in a directed graph:

1. Run the Floyd-Warshall algorithm on the graph.
2. For each vertex in the graph, check if the distance between the vertex and itself is less than infinity. If it is, then there is a cycle that starts and ends at that vertex. The length of the shortest cycle is equal to the distance between the vertex and itself.
3. Return the shortest cycle length, or -1 if there is no cycle.

Question 4:

The required algorithm would be as follows:

1. Use Dijkstra's algorithm to find the shortest path lengths from ***A*** to all other vertices in the graph. This is stored in a dictionary called ***distToA***.
2. Reverse the graph and use Dijkstra's algorithm to find the shortest path lengths from all other vertices in the graph to ***A***. This is stored in a dictionary called ***distFromA***.
3. For each pair of vertices, ***i*** and ***j***, the shortest path length that passes through ***A*** is calculated as ***distToA[i]*** ***+ distFromA[j].***

The algorithm takes ***O((|V| + |E|) log |V|)*** time, where **|V|** is the number of vertices in the graph and **|E|** is the number of edges in the graph.