Prims Algorithm:

It is a greedy algorithm. It starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST; the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

If there is no directed edge from explored nodes of the MST to the remaining unexplored nodes, the algorithm gets stuck even though there are edges from unexplored nodes to explored nodes in the MST.

Prim’s algorithm assumes that all vertices are connected. But in a directed graph, every node is not reachable from every other node. So, Prim’s algorithm fails due to this reason.

Since our required graph is a directed graph, the prim’s algorithm will not work for this.

Kruskal’s Algorithm:

It sorts all the edges in non-decreasing order of their weight. Then it picks the smallest edge. The algorithm then checks if it forms a cycle with the spanning tree formed so far. If cycle is not formed, this edge is included. Else, it is discarded. The second step is repeated until there are (V-1) edges in the spanning tree.

In Kruskal’s algorithm, in each step, it is checked that if the edges form a cycle with the spanning-tree formed so far. But Kruskal’s algorithm fails to detect the cycles in a directed graph as there are cases when there is no cycle between the vertices but Kruskal’s Algorithm assumes it to cycle and don’t take consider some edges due to which Kruskal’s Algorithm fails for directed graph.

Dijkstra’s Algorithm

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph. It differs from the minimum spanning tree because the shortest distance between two vertices might not include all the vertices of the graph.

Dijkstra's Algorithm works on the basis that any subpath of the shortest path between vertices A and D is also the shortest path between vertices B and D. Dijkstra used this property in the opposite direction i.e., we overestimate the distance of each vertex from the starting vertex. Then we visit each node and its neighbours to find the shortest subpath to those neighbours.

The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

The problem with Dijkstra’s algorithm is that it is believed that all costs in the given graph are non-negative, so adding any positive number on a vertex that has already been visited will never change its minimality. Since Dijkstra follows a Greedy Approach, once a node is marked as visited it cannot be reconsidered even if there is another path with less cost or distance. This issue arises only if there exists a negative weight or edge in the graph. So, this algorithm fails to find the minimum distance in case of negative weights.

Solution:

Since all 3 are unable to handle a directed, negative weighted graph, we need to find an alternate algorithm to find the minimum path from vertex A. This problem is well handled by the [Bellman-Ford algorithm](https://pencilprogrammer.com/algorithms/bellman-ford-algorithm/). The Bellman–Ford algorithm is an [algorithm](https://en.wikipedia.org/wiki/Algorithm) that computes [shortest paths](https://en.wikipedia.org/wiki/Shortest_path) from a single source [vertex](https://en.wikipedia.org/wiki/Vertex_(graph_theory)) to all of the other vertices in a [weighted digraph](https://en.wikipedia.org/wiki/Weighted_digraph). It is slower than [Dijkstra's algorithm](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm) for the same problem, but more versatile, as it is capable of handling graphs in which some of the edge weights are negative numbers.

Assuming there are negative edges in the graph, but there are no negative cycles, the Bellman-Ford algorithm iterates through all the edges multiple times (V-1 times), irrespective of the fact whether the vertices are visited or not, thus resulting in successfully finding the optimal low-cost path.

Reasoning for using C Programming language:

I’ve used C programming language to code all 3 algorithms as it is the language I used previously to code these algorithms and I found it easier to understand.