Lab Sheet – 3

Graph Algorithms in Real – Life Applications

Problem 1: Social Network Friend Suggestion

- **Time Complexity:**

The algorithm's time complexity is **O(V + E)**, where V is the number of users (vertices) and E is the number of friendships (edges).

- To find "friends of friends," we first iterate through the target user's direct friends (let's say they have *k* friends).
- Then, we iterate through the friends of each of those *k* friends.
- In the worst case, this traversal resembles a Breadth-First Search (BFS) or Depth-First Search (DFS) that extends two levels out from the user. The complexity of a full graph traversal is O(V+E), and this operation is a subset of that, making its upper bound O(V+E).

- **Scalability for Large Networks:**

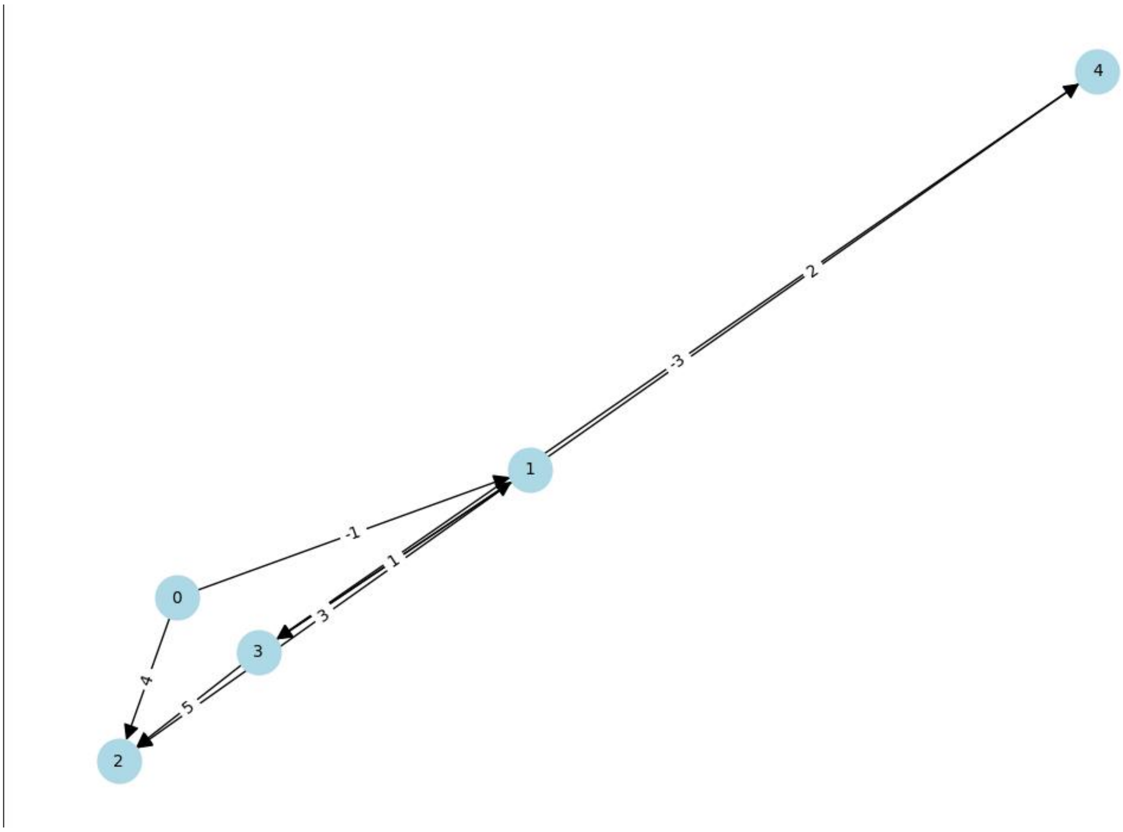
- For a massive network like LinkedIn or Facebook, which has billions of users and edges, performing an O(V+E) traversal in real-time every time a user requests suggestions is not feasible. The query would be too slow.
- Real-world systems solve this scalability problem by **pre-computation**. Friend suggestions are calculated offline (e.g., during off-peak hours) using distributed graph processing frameworks.
- The results are then stored in a database or cache. When you visit your profile, the application simply performs a quick lookup to retrieve this pre-computed list of suggestions, which feels instantaneous to you.

```
Finding friend suggestions for: Alice

Graph Adjacency List: {'Alice': {'Carol', 'Bob'}, 'Bob': {'Eve', 'David', 'Alice'}, 'Carol': {'David', 'Alice', 'Fiona'},
Direct friends for Alice: ['Carol', 'Bob']
Suggested friends for Alice: ['Eve', 'David', 'Fiona']

Execution Time: 0.000084639 seconds
```

Problem 2: Route Finding on Google Maps



➢ Preference for Negative Weights:

Bellman-Ford is preferred over Dijkstra's algorithm in graphs with negative edge weights because Dijkstra's greedy approach fails in this scenario. Dijkstra's assumes that once a path to a node is finalized, it's the shortest possible path. However, a negative edge encountered later could create a "shortcut" that violates this assumption, leading to an incorrect result. Bellman-Ford's iterative approach relaxes all edges V-1 times, which guarantees it finds the correct shortest path even if negative weights are present. It can also detect negative weight cycles2, which Dijkstra's cannot.

- Time Complexity O(V * E):

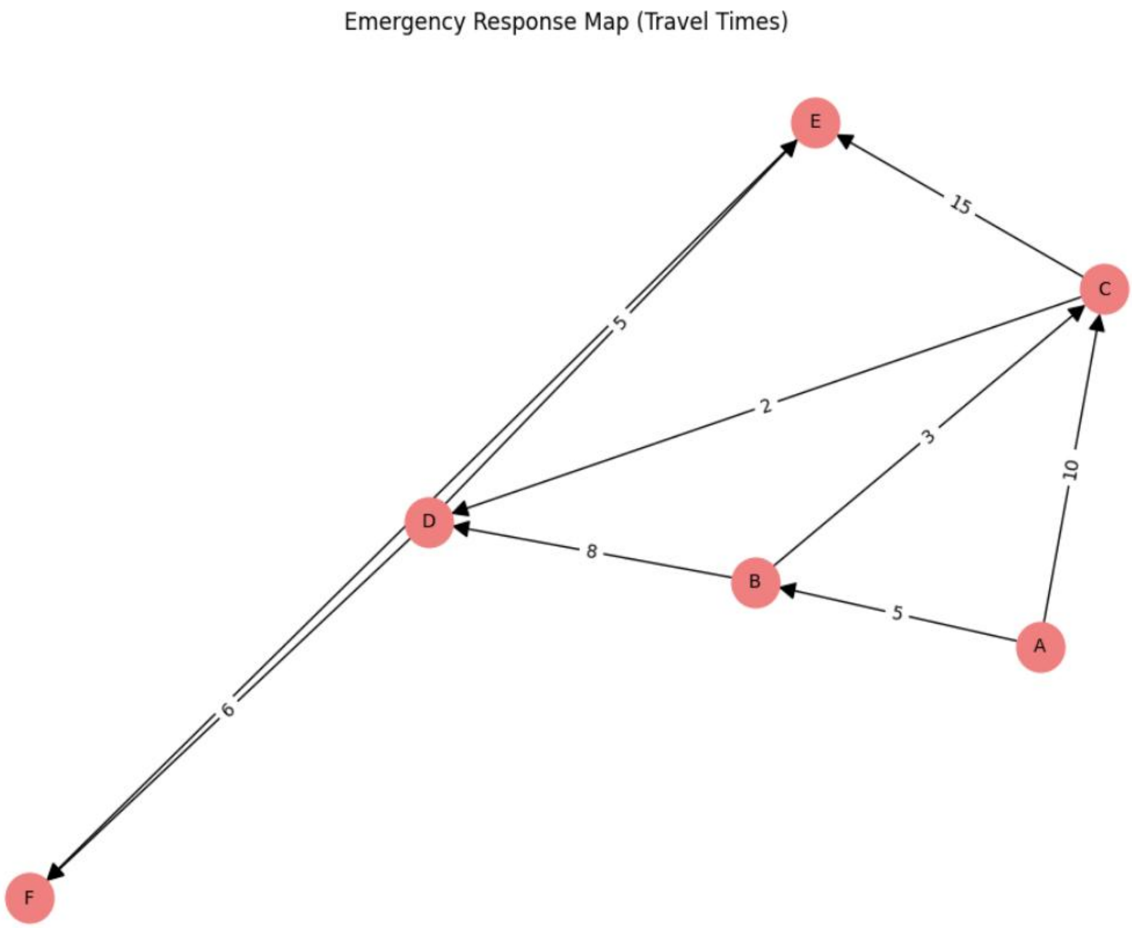
The time complexity of the Bellman-Ford algorithm is O(V*E), where V is the number of vertices (locations) and E is the number of edges (roads).

- The algorithm's core is a main loop that iterates **V-1 times**.
- Inside this main loop, it iterates through **every edge (E)** in the graph to "relax" it (i.e., check if a shorter path can be found via that edge).
- This results in a complexity of (V-1) X E.
- An additional pass over all E edges is performed to detect negative weight cycles[4].
- Therefore, the total time complexity simplifies to **O(V*E)**.

```
--- Bellman-Ford Results ---
Shortest distances from source node 0:
  Node 0: 0
  Node 1: -1
  Node 2: 2
  Node 3: -2
  Node 4: 1

Execution Time: 0.000983953 seconds
```

Problem 3: Emergency Response System



- ➢ Time Complexity: O(E log V) using min-heap

The time complexity for Dijkstra's algorithm, when implemented efficiently using a min-heap (priority queue), is O(E log V).

- V is the number of vertices (intersections), and E is the number of edges (roads).
- Every vertex is added to the min-heap once. Extracting the minimum-distance vertex from the heap takes O(log V) time.
- We perform this extraction V times.
- For every edge, we might perform a "decrease-key" operation (updating the distance), which also takes O(log V) time.

- This gives a total complexity of O(V log V + E log V), which simplifies to **O(E log V)** in a connected graph (where E is typically greater than or equal to V).

➢ Why Dijkstra is Unsuitable for Negative Weights.

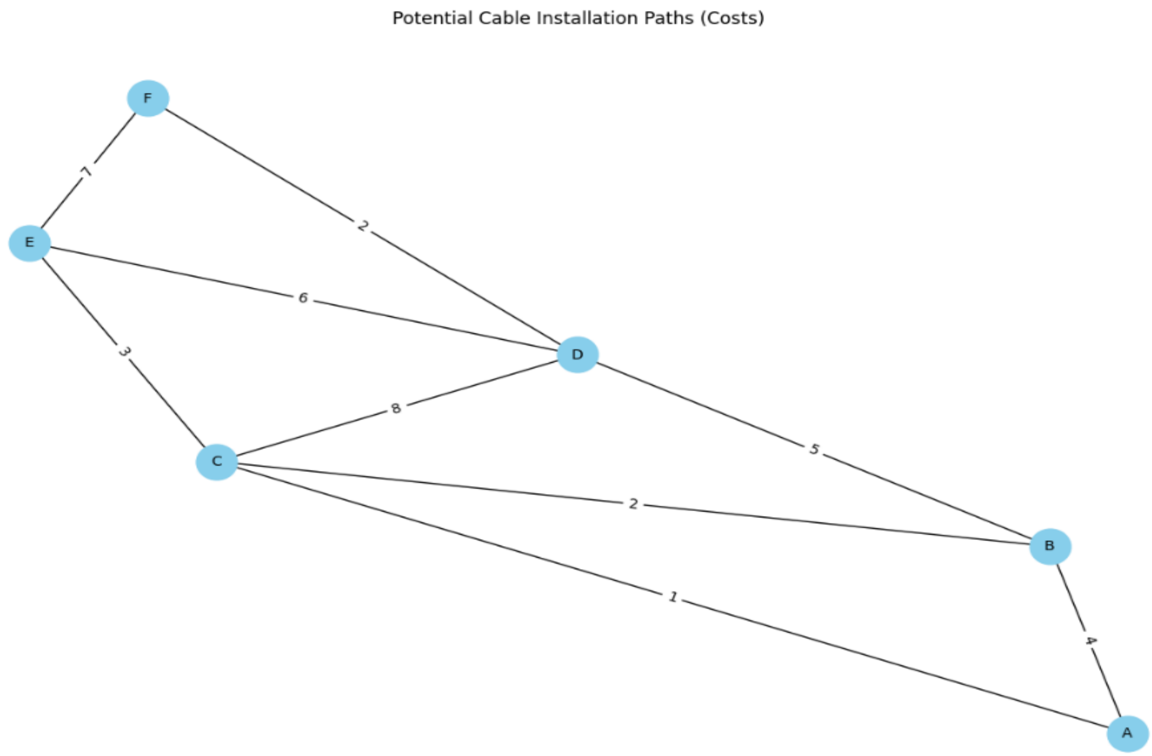
Dijkstra's algorithm is unsuitable for graphs with negative edge weights because its core "greedy" strategy fails:-

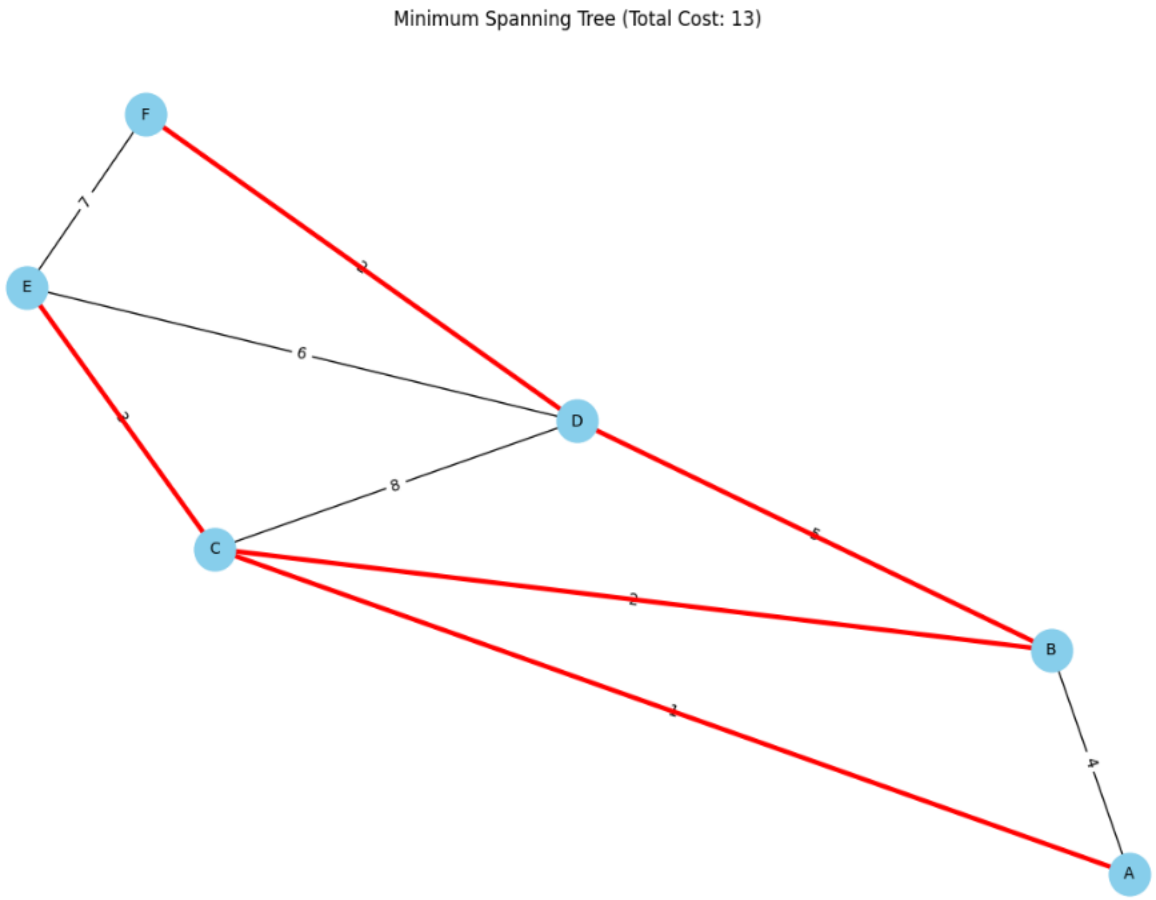
- Dijkstra's operates on the assumption that once it finalizes the shortest path to a vertex, no shorter path can ever be found.
- A **negative edge** breaks this rule. A path that looks longer might later connect to a negative edge, creating a "shortcut" that results in a shorter overall distance.
- Because Dijkstra's finalizes paths greedily without looking ahead for potential negative shortcuts, it will fail to find the correct shortest path in such cases. Bellman-Ford (Problem 2) is used instead as it correctly handles negative weights.

```
...
--- Dijkstra's Algorithm Results ---
Fastest travel times from A:
  To A: 0
  To B: 5
  To C: 8
  To D: 10
  To E: 15
  To F: 16

Execution Time: 0.000155449 seconds
```

Problem 4: Network Cable Installation





- ➢ Complexity Comparison: Prim’s vs. Kruskal’s [1]

- Prim's Algorithm (using a min-heap): The time complexity is O(E \log V). This is because every edge (E) is processed, and heap operations (like adding an edge or extracting the minimum edge) take O(log V) time.
- Kruskal's Algorithm (using Union-Find): The time complexity is O(E log E). This is dominated by the initial step of sorting all E edges by weight. The subsequent Union-Find operations are very fast (nearly constant time on average).
- Comparison: Prim's is generally faster for dense graphs (where E is close to V^2), while Kruskal's is often faster for sparse graphs (where E is much smaller than V^2).

- ➢ Applicability in Infrastructure Cost Optimization

Minimum Spanning Tree (MST) algorithms are directly applicable to infrastructure cost optimization. This problem is a classic example: finding the minimum total length of cable needed to connect all offices. This same logic applies to many real-world scenarios, such as:

- Telecom & IT: Connecting all nodes in a computer network with the least amount of fiber optic cable[4].
- Power Grids: Designing the cheapest way to connect all cities or substations to a power source.
- Transportation: Building the minimum length of road or railway required to connect a set of towns.

```
--- Prim's Algorithm MST Results ---
Total minimum cost to connect all offices: 13

Edges selected in MST:
  A -- C (Cost: 1)
  C -- B (Cost: 2)
  C -- E (Cost: 3)
  B -- D (Cost: 5)
  D -- F (Cost: 2)

Execution Time: 0.000120401 seconds
```

Problem	**Graph Algorithm**	**Time Complexity**	**Application Domain**
Social Network Suggestion	BFS / DFS	O(V + E)	Social Media
Google Maps Routing	Bellman-Ford	O(VE)	Navigation
Emergency Path Planning	Dijkstra’s	O(E log V)	Disaster Response
Cable Installation	MST (Prim/Kruskal)	O(E log V)	Infrastructure