	School of Engineering & Technology	
	Department: SOET	**Session: 2025-26**
	Program: BCA (AI&DS)	**Semester: V**
	Course Code: ENCA351	**Number of students: 188**
	Course Name: Design and Analysis of Algorithms Lab	**Faculty: Dr. Aarti**

Lab Assignment -3: Graph Algorithms in Real-Life Applications

Instructions:

- Assignment must be submitted within the deadline communicated by the instructor at the time of release.
- Assignment must be submitted on https://lms.krmangalam.edu.in/
- You must provide a link to your GitHub repository with your submission on LMS.
- Use of ChatGPT and similar tools is strictly prohibited.
- The assignment needs to be submitted by each individual.
- This assignment carries a total of 10 marks.
- Assignment will be assessed based on the evaluation rubrics.
- Estimated Duration: 10-12 hours

Assignment Title: Solving Real-World Problems Using Graph Algorithms

---

Real-World Problem Context

Graph algorithms are at the heart of numerous real-life applications ranging from social networking and navigation to disaster response and infrastructure development. This project guides you to implement and analyze four practical problems using fundamental graph traversal and optimization strategies.

Learning Objectives

By completing this project, you will:

- Implement core graph algorithms such as BFS, DFS, Dijkstra, Bellman-Ford, and MST.

- Understand the mapping of real-world problems to graph structures.
- Analyze time complexity and performance impact through practical testing.
- Communicate your design, results, and visualizations clearly through code and documentation.

Assignment Tasks

Problem 1: Social Network Friend Suggestion

Graph Algorithm: BFS / DFS

Real-World Application: Facebook, LinkedIn

Objective: Suggest new connections based on mutual friends.

Sub-Tasks and What to Do

1. **Graph Modeling**
   - Represent users as nodes and friendships as edges in an undirected graph.
   - Use an adjacency list for efficient storage and traversal.
2. **Algorithm Design**
   - Perform BFS or DFS starting from the given user.
   - Identify "friends of friends" who are not already connected to the user.
3. **Input**
   - Sample graph connections (e.g., A–B, A–C, B–D, etc.)
4. **Output**
   - List of suggested friends for a user.
5. **Analysis**
   - Discuss time complexity of BFS/DFS traversal (O(V + E)).
   - Comment on scalability for large networks.

Problem 2: Route Finding on Google Maps

Graph Algorithm: Bellman-Ford

Real-World Application: Navigation systems

Objective: Find shortest path from source to all nodes, even if some edge weights are negative.

Sub-Tasks and What to Do

2. **Graph Modeling**

- Represent cities/locations as nodes and roads as directed edges with weights.
- Include some negative weights to test the robustness of Bellman-Ford.

3. **Algorithm Design** ○ Implement Bellman-Ford

   algorithm for shortest path computation.

   - Detect negative weight cycles and handle them gracefully.

4. **Input**
   - List of edges with weights: (source, destination, weight)
5. **Output**
   - Distance array showing minimum distance from source to all vertices.
6. **Analysis** ○ Explain why Bellman-Ford is

   preferred in graphs with negative weights. ○

   Discuss time complexity O(V * E).

Problem 3: Emergency Response System

Graph Algorithm: Dijkstra's Algorithm

Real-World Application: Disaster Management

Objective: Identify the fastest route for emergency vehicles in a weighted city map (all weights positive).

Sub-Tasks and What to Do

1. **Graph Modeling**
   - Represent intersections as nodes and roads with travel times as weighted edges.
   - Use an adjacency list for storing graph data.
2. **Algorithm Design**
   - Use a priority queue (min-heap) for implementing Dijkstra's algorithm efficiently. ○ Keep updating distances until the shortest path to all reachable nodes is found.
3. **Input**
   - Dictionary or list of edges with weights between intersections.
4. **Output**

    - Shortest distances from the source node to all others.

5. **Analysis**
    - Time complexity: O(E log V) using min-heap.
    - Comment on why Dijkstra is unsuitable for graphs with negative weights.

Problem 4: Network Cable Installation

Graph Algorithm: Minimum Spanning Tree (Prim's Algorithm or Kruskal's)

Real-World Application: Telecom & IT Infrastructure

Objective: Connect all offices/nodes with the minimum total length of cable.

Sub-Tasks and What to Do

1. **Graph Modeling**
    - Nodes represent office buildings.
    - Edges represent possible cable paths with associated costs (weights).
2. **Algorithm Design**
    - Use **Prim's algorithm** with a priority queue to construct MST.
    - Alternatively, implement **Kruskal's algorithm** using Union-Find.
3. **Input**
    - Undirected weighted graph in adjacency list format.
4. **Output**
    - Total minimum cost to connect all nodes (sum of MST edges).
    - Optional: list of edges selected in MST.
5. **Analysis**
    - Compare Prim's and Kruskal's complexities.
    - Comment on applicability in infrastructure cost optimization.

Task 3: Experimental Profiling & Visualization For each graph algorithm:

- Use time module to measure execution time.
- Use memory_profiler to track memory usage.
- Visualize execution time vs. number of nodes/edges (optional).

- Comment on time complexity and practical performance impact.

Task 4: Final Summary and Documentation
Include the following:

Problem	Graph Algorithm	Time Complexity	Application Domain	Notes
Social Network Suggestion	BFS / DFS	O(V + E)	Social Media	Suggest mutual friends
Google Maps Routing	Bellman-Ford	O(VE)	Navigation	Works with negative weights
Emergency Path Planning	Dijkstra’s	O(E log V)	Disaster Response	Fastest path in a positive-weighted map
Cable Installation	MST (Prim/Kruskal)	O(E log V)	Infrastructure	Minimum cable cost

- Add reflections: How did real-world context influence algorithm choice?
- Include your GitHub README.md with:
    - Problem overview ○ Instructions to run code
    - External references and acknowledgments

Evaluation Rubric (Total: 10 Marks)

Criteria	Marks	Description
GitHub Setup & Organization	1	README, repo structure, environment setup
Algorithm Implementation	3	Correct Python implementation of all 4 graph problems
Profiling & Visualization	2.5	Time/memory measurement and plotting (optional)
Analysis & Discussion	2.5	Complexity discussion, algorithm choice justification
Code Quality & Documentation	1	Commented code, markdown explanations, citations

---

Submission Instructions

- Push to GitHub:
    - README.md: overview and usage

- graph_realworld.ipynb: full implementation
- requirements.txt: list of packages
- Optional: images/ folder for plots

- Submit your **GitHub repo link** via LMS.

Academic Integrity & Plagiarism Policy

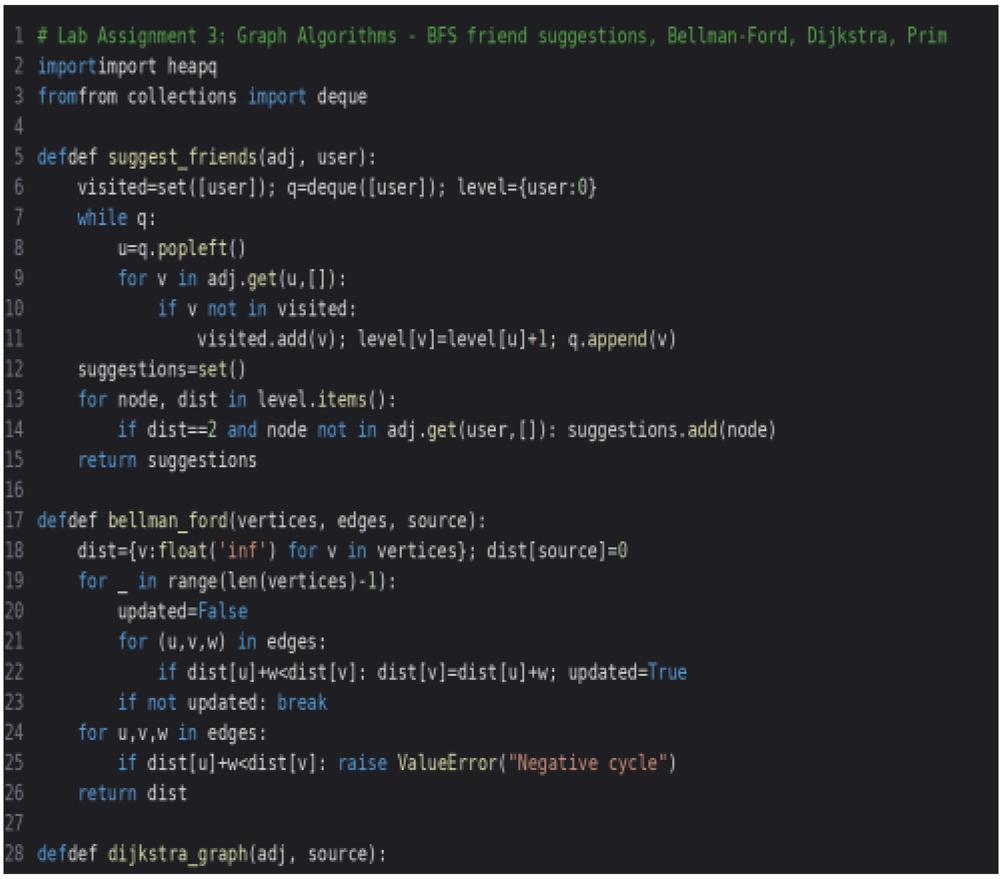
- Individual submission
- Plagiarism will result in **zero marks**
- Properly **cite** external sources

Support Resources

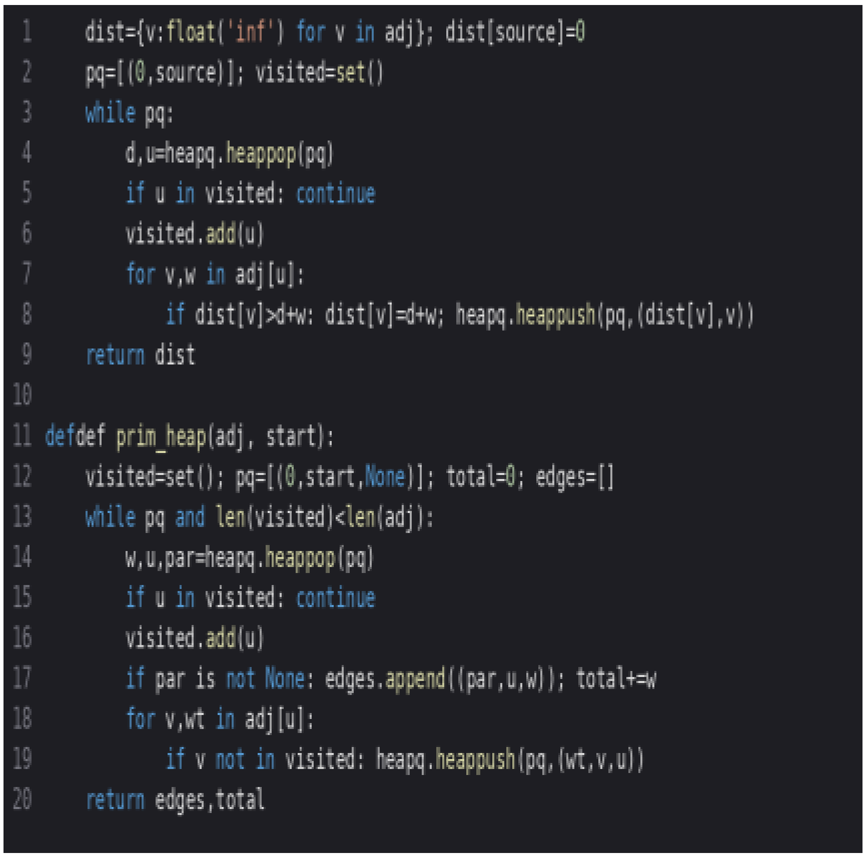
- CLRS – *Introduction to Algorithms*
- Python Graph Libraries (e.g., networkx optional)
- Profiling tools: time, memory_profiler
- Visualization: matplotlib

Contact: Dr. Aarti, aarti.sangwan@krmangalam.edu.in

Solution-:

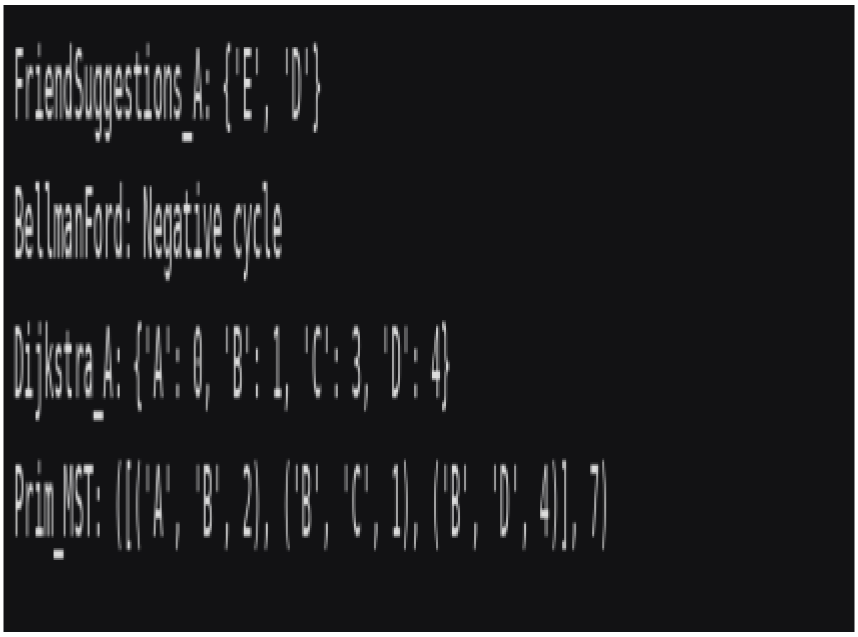


```python
# Lab Assignment 3: Graph Algorithms - BFS friend suggestions, Bellman-Ford, Dijkstra, Prim
importimport heapq
fromfrom collections import deque

defdef suggest_friends(adj, user):
    visited=set([user]); q=deque([user]); level={user:0}
    while q:
        u=q.popleft()
        for v in adj.get(u,[]):
            if v not in visited:
                visited.add(v); level[v]=level[u]+1; q.append(v)
    suggestions=set()
    for node, dist in level.items():
        if dist==2 and node not in adj.get(user,[]): suggestions.add(node)
    return suggestions

defdef bellman_ford(vertices, edges, source):
    dist={v:float('inf') for v in vertices}; dist[source]=0
    for _ in range(len(vertices)-1):
        updated=False
        for (u,v,w) in edges:
            if dist[u]+w<dist[v]: dist[v]=dist[u]+w; updated=True
        if not updated: break
    for u,v,w in edges:
        if dist[u]+w<dist[v]: raise ValueError("Negative cycle")
    return dist

defdef dijkstra_graph(adj, source):
```



```python
    dist={v:float('inf') for v in adj}; dist[source]=0
    pq=[(0,source)]; visited=set()
    while pq:
        d,u=heapq.heappop(pq)
        if u in visited: continue
        visited.add(u)
        for v,w in adj[u]:
            if dist[v]>d+w: dist[v]=d+w; heapq.heappush(pq,(dist[v],v))
    return dist

defdef prim_heap(adj, start):
    visited=set(); pq=[(0,start,None)]; total=0; edges=[]
    while pq and len(visited)<len(adj):
        w,u,par=heapq.heappop(pq)
        if u in visited: continue
        visited.add(u)
        if par is not None: edges.append((par,u,w)); total+=w
        for v,wt in adj[u]:
            if v not in visited: heapq.heappush(pq,(wt,v,u))
    return edges,total
```

Output-:



```
FriendSuggestions_A: {'E', 'D'}

BellmanFord: Negative cycle

Dijkstra_A: {'A': 0, 'B': 1, 'C': 3, 'D': 4}

Prim_MST: ([('A', 'B', 2), ('B', 'C', 1), ('B', 'D', 4)], 7)
```