ECON381 FALL 2024

Homework Assignment 4

Question 1: Explain what a minimum spanning tree (MST) is and then create an MST fort he above graph. You can use Kruskal's algorithm or any similar algorithm. Show all your steps.

Answer:

1- What is minimum spanning tree (MST)?
A minimum spanning tree is a subgraph of an undirected, connected graph that includes all the vertices of the graph, is a tree and has the minimum possible total edge weight.

2- Creating an MST graph by using Kruskal's algorithm

- Short all the edges in ascending order of their weights.
- Start with an empty set of edges fort he MST.
- Add edges to the MST one at a time, ensuring no cycles are formed.
- Stop when the MST contains *n - 1* edges, where *n* is the number of vertices.

Vertices: *V = {A, B, C, D, E}*

Edges with weights:

- *A - B : 1*
- *A - C : 5*
- *B - C : 4*
- *B - D : 3*
- *C - D : 2*
- *D - E : 7*
- *C - E : 6*

Step 1: Sorting

{(A−B,1), (C−D,2), (B−D,3), (B−C,4), (A−C,5), (C−E,6), (D−E,7)}

Step 2: Initialize an empty MST and add edges without forming cycles.

- Start with empty MST: MST={}MST = \{\}MST={}.

Step-by-Step Process:

- Add A−BA-BA−B (weight 1). No cycle is formed.
  MST={(A−B)}
- Add C−DC-DC−D (weight 2). No cycle is formed.
  MST={(A−B),(C−D)}
- Add B−DB-DB−D (weight 3). No cycle is formed.
  MST={(A−B),(C−D),(B−D)}
- Add B−CB-CB−C (weight 4). No cycle is formed.
  MST={(A−B),(C−D),(B−D),(B−C)}

  Stop here because we now have $n - 1 = 5 - 1 = 4n - 1 = 5 - 1 = 4n - 1 = 5 - 1 = 4$ edges.

Step 3: Result

The edges in the MST are:

MST = {(A−B,1), (C−D,2), (B−D,3), (B−C,4)}

The total weight of the MST is: 1 + 2 + 3 + 4 = 10

Question 2: We need to know if the MST you created is unique or not. Do some research on the conditions for uniqueness of a generated MST. Is the above graph suitable for guaranteed uniqueness?

-Is the Minimum Spanning Tree (MST) Unique?

To determine whether the MST is unique, we need to analyze the **conditions for MST** uniqueness.

Conditions for a Unique MST

1. **Distinct Edge Weights:**
   - A graph will have a unique MST if all the edge weights are distinct. This is because the algorithms (like Kruskal's or Prim's) always select the smallest edge, and no ties can occur during the selection process.
2. **Tie Cases:**
   - If there are duplicate edge weights, it is possible for multiple valid MSTs to exist. In such cases, different edges with the same weight may be chosen at different steps, leading to different MST structures.

Analyzing the Above Graph

The given graph has the following edges and weights:

{(A−B,1), (C−D,2), (B−D,3), (B−C,4), (A−C,5), (C−E,6), (D−E,7)}

1. **Are the edge weights distinct?**
   - Yes, all the edge weights are distinct (1,2,3,4,5,6,71, 2, 3, 4, 5, 6, 7).
   - This means there are no ties when selecting edges during the MST construction.
2. **Can the graph guarantee a unique MST?**
   - Since all edge weights are distinct, the MST generated using Kruskal's or Prim's Algorithm is guaranteed to be unique.

Question 3: Calculate the shortest paths from node A to all other nodes using Dijkstra's algorithm. Show all your steps.

Answer: We use the same graph as before:

Vertices: V={A, B, C, D, E}
Edges with weights:

- A – B : 1
- A – C : 5
- B – C : 4
- B – D : 3
- C – D : 2
- D – E : 7
- C – E : 6

Step 1: Initialize Tables

1. **Distance Table**: Set all distances to ∞ (representing unreachable nodes) except the source A, which is 0.

   ```
   Distances: {A: 0, B: ∞, C: ∞, D: ∞, E: ∞}
   ```

2. **Visited Set**: Initially empty.

   ```
   Visited: {}
   ```

3. **Priority Queue**: Start with (A,0) (the source and its distance).

   ```
   Priority Queue: [(A, 0)]
   ```

Step 2: Algorithm Execution

We process the graph step by step, updating distances and the priority queue.

Iteration 1: Process Node A

1. Node A is removed from the priority queue:

   ```
   Priority Queue: []
   Current Node: A (Distance: 0)
   ```

2. Update distances to A's neighbors:
   - A → B : Distance = 0 + 1 = 1
   - A → C : Distance = 0 + 5 = 5

   ```
   Distances: {A: 0, B: 1, C: 5, D: ∞, E: ∞}
   ```

3. Add neighbors to the priority queue:

   ```
   Priority Queue: [(B, 1), (C, 5)]
   ```

4. Mark A as visited:

```
Visited: {A}
```

Iteration 2: Process Node B

1. Node B is removed from the priority queue:

```
Priority Queue: [(C, 5)]
Current Node: B (Distance: 1)
```

2. Update distances to B's neighbors:
   - B → C : Distance = 1 + 4 = 5 (already 5, no change).
   - B → D : Distance = 1 + 3 = 4

```
Distances: {A: 0, B: 1, C: 5, D: 4, E: ∞}
```

3. Add D to the priority queue:

```
Priority Queue: [(C, 5), (D, 4)]
```

4. Mark B as visited:

```
Visited: {A, B}
```

Iteration 3: Process Node D

1. Node D is removed from the priority queue:

```
Priority Queue: [(C, 5)]
Current Node: D (Distance: 4)
```

2. Update distances to D's neighbors:
   - D → E : Distance = 4 + 7 = 11

```
Distances: {A: 0, B: 1, C: 5, D: 4, E: 11}
```

3. Add E to the priority queue:

```
Priority Queue: [(C, 5), (E, 11)]
```

4. Mark D as visited:

```
Visited: {A, B, D}
```

Iteration 4: Process Node C

1. Node C is removed from the priority queue:

```
Priority Queue: [(E, 11)]
Current Node: C (Distance: 5)
```

2. Update distances to C's neighbors:
    - C → E : Distance = 5 + 6 = 11 (already 11, no change).

```
Distances: {A: 0, B: 1, C: 5, D: 4, E: 11}
```

3. Mark C as visited:

```
Visited: {A, B, C, D}
```

Iteration 5: Process Node E

1. Node E is removed from the priority queue:

```
Priority Queue: []
Current Node: E (Distance: 11)
```

2. E has no unvisited neighbors. No updates.
3. Mark E as visited:

```
Visited: {A, B, C, D, E}
```

Step 3: Final Distances

The shortest distances from A to all other nodes are:

Distances: {A:0, B:1, C:5, D:4, E:11}

Step 4: Path Reconstruction (Optional)

If we want to find the exact paths, we would store the "previous node" for each vertex during updates. For example:

- Path to C : A → C
- Path to E : A → B → D → E

Question 4: Explain what a critical edge in a graph is. Then try to find critical edges in this graph. Show detailed steps for a single edge removal. If you can not find any critical edge, explain why.

Answer: **What is a Critical Edge?**

A **critical edge** (also known as a bridge) in a graph is an edge that, when removed, **increases the number of connected components** of the graph. In other words:

- If you remove a critical edge, the graph becomes disconnected.
- These edges are essential for maintaining the connectivity of the graph.

Steps to Find Critical Edges

To identify a critical edge, follow these steps:

1. **Remove an edge from the graph.**
2. **Check if the graph becomes disconnected:**
    - Perform a Depth-First Search (DFS) or Breadth-First Search (BFS) starting from any node.
    - If not all nodes are reachable, the edge is critical.

Analyzing the Graph

The graph is as follows:

Vertices: V = { A, B, C, D, E}
Edges with weights:

- A−B:1
- A−C:5
- B−C:4
- B−D:3
- C−D:2
- D−E:7
- C−E:6

Single Edge Removal Analysis

We'll test a single edge, A−B, to determine if it's critical.

Step 1: Remove Edge A−B

The graph now looks like this:

- Remaining edges: A−C, B−C, B−D, C−D, D−E, C−E

Step 2: Check Connectivity

Perform a DFS or BFS starting from node A. The traversal proceeds as follows:

- From A, we can reach C (via A−C).
- From C, we can reach B, D, and E (via C−B, C−D, C−E).

Since all nodes are reachable, the graph remains connected. Therefore, A−B is **not a critical edge**.

Repeating for Other Edges

Edge C−D:

1. Remove C−D. Remaining edges: A−B, A−C, B−C, B−D, C−E, D−E.
2. Start DFS or BFS:
    - From A, we reach C (via A−C).
    - From C, we can reach all other nodes via C−B, C−E, and so on.
3. Result: The graph remains connected. C−D is **not critical**.

Edge D−E:

1. Remove D−E. Remaining edges: A−B,A−C,B−C,B−D,C−D,C−E
2. Start DFS or BFS:
    - From A, we can reach D (via A−C,B−C,C−D).
    - From D, we can no longer reach E.
3. Result: The graph becomes disconnected. D−E is a **critical edge**.

Critical Edges in the Graph

By applying this method to all edges, we find the following:

- **Critical edge:** D−E.
- **Non-critical edges:** A−B,A−C,B−C,B−D,C−D,C−E.

Why are Most Edges Not Critical?

This graph has **multiple paths** between most pairs of nodes. For example:

- Even if A−B is removed, A and B are still connected via A−C−B.
- Similarly, C and D remain connected via C−B−D.

However, D−E is **critical** because E is only connected to the rest of the graph through D.

Question 5: Explain what an articulation point in a graph is. Then try to find articulation points in this graph. Show detailed steps for a single vertex removal. If you can not find any articulation point, explain why.

Answer: An articulation point in a graph is a vertex that, when removed along with all of its incident edges, increases the number of connected components in the graph**.**

- **Remove each vertex one at a time**, and check connectivity:

- Remove A: Disconnects B and C from the rest of the graph. **A is an articulation point.**
- Remove B: Disconnects C from A, D, E, F. **B is an articulation point.**
- Remove C: Graph remains connected. **C is not an articulation point.**
- Remove D: Graph remains connected. **D is not an articulation point.**
- Remove E: Disconnects C and F. **E is an articulation point.**
- Remove F: Graph remains connected. **F is not an articulation point.**

Question 6: Suppose you want to go from A to E, the path you are given is A-B-C-E (based on Dijkstra's algorithm). You are at B and learn that C is now unavailable. Given that you know there are no critical edges or articulation points beforehand, can you be sure that there is now another path towards E without any calculations?

Answer: Yes, you can be sure that there is another path to E without any calculations because:

- **No critical edges**: A critical edge is an edge whose removal would disconnect the graph or make it impossible to traverse between certain vertices. Since the graph has no critical edges, the removal of edge B−C (or any edge involving C) will not make E unreachable.
- **No articulation points**: An articulation point is a vertex whose removal increases the number of connected components. Since C is not an articulation point, its removal will not disconnect E from the rest of the graph.

Thus, even though C is unavailable, the graph remains connected, and there must be an alternate path from A to E.

Question 7: Do some research on the concept of graph robustness, and explain it using critical edges and articulation points.

Answer: Graph robustness refers to a network's ability to maintain its overall connectivity and functionality despite failures or targeted attacks on its components. In graph theory, two critical elements that influence robustness are **articulation points** and **bridges** (also known as critical edges).

Articulation Points: An articulation point, or cut vertex, is a node whose removal increases the number of connected components in the graph. This means that the node is a critical connector within the network, and its removal would fragment the network into disjoint subgraphs. Identifying articulation points is essential for understanding vulnerabilities in network structures.

Bridges (Critical Edges): A bridge, or cut-edge, is an edge in a graph whose removal increases the number of connected components. In other words, a bridge is a single connection that, if severed, would disconnect parts of the network, indicating a critical link in maintaining the network's integrity.

The presence of articulation points and bridges indicates potential weaknesses in a network's structure. Networks lacking these critical nodes and edges are termed **biconnected** and are generally more robust, as they do not have single points of failure. Understanding and identifying these elements help in designing resilient networks and in developing strategies to mitigate the impact of component failures.

In summary, analyzing articulation points and bridges within a graph provides valuable insights into its robustness, highlighting areas that may require reinforcement to prevent network fragmentation.