1. Explain what a Minimum Spanning Tree (MST) is and create an MST for the graph.

What is a Minimum Spanning Tree (MST)?

- An MST is a subset of edges in a connected, weighted graph such that:
  - All vertices are connected (it spans the graph).
  - The total weight of the edges is minimized.
  - There are no cycles.

Steps to Create an MST Using Kruskal's Algorithm

Step 1: Sort All Edges by Weight

- Arrange all edges in ascending order of their weights. This ensures that the edges with the smallest weights are processed first.

Example Edge List (Weight in parentheses):

- *(A,B,1)(A, B, 1)*(A,B,1), *(B,C,2)(B, C, 2)*(B,C,2), *(A,C,3)(A, C, 3)*(A,C,3), *(C,D,4)(C, D, 4)*(C,D,4), *(D,E,5)(D, E, 5)*(D,E,5), *(B,E,6)(B, E, 6)*(B,E,6)

Step 2: Initialize Disjoint Sets

- Use the **Union-Find** data structure to track connected components. Initially, each vertex is its own set (disjoint).

Example:

- Initially, sets are: *{A},{B},{C},{D},{E}\{A\}, \{B\}, \{C\}, \{D\}, \{E\}*{A},{B},{C},{D},{E}

Step 3: Add Edges to the MST

- Process edges in the sorted order.
- Add an edge to the MST if it does not form a cycle.
- Use the Union-Find algorithm to determine if two vertices are in the same set (cycle detection).
- Stop when the MST contains *V−1V - 1*V−1 edges (where *VV* is the number of vertices).

Graphical Example

Initial Graph:

```
Vertices: A, B, C, D, E
Edges (with weights):
  A --(1)-- B
  B --(2)-- C
  A --(3)-- C
  C --(4)-- D
  D --(5)-- E
  B --(6)-- E
```

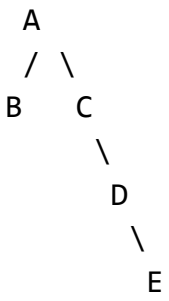
Step-by-Step Execution

1. **Sorted Edges:**
   a. *(A,B,1)(A, B, 1)*(A,B,1), *(B,C,2)(B, C, 2)*(B,C,2), *(A,C,3)(A, C, 3)*(A,C,3), *(C,D,4)(C, D, 4)*(C,D,4), *(D,E,5)(D, E, 5)*(D,E,5), *(B,E,6)(B, E, 6)*(B,E,6)
2. **Edge Selection (Iterate through sorted edges):**
   a. **Edge *(A,B,1)(A, B, 1)*(A,B,1):**
      i. Add to MST (no cycle).
      ii. Current MST: *(A,B,1)(A, B, 1)*(A,B,1)
      iii. Sets: *{A,B},{C},{D},{E}|{A, B|}, |{C|}, |{D|}, |{E|}*{A,B},{C},{D},{E}
   b. **Edge *(B,C,2)(B, C, 2)*(B,C,2):**
      i. Add to MST (no cycle).
      ii. Current MST: *(A,B,1),(B,C,2)(A, B, 1), (B, C, 2)*(A,B,1),(B,C,2)
      iii. Sets: *{A,B,C},{D},{E}|{A, B, C|}, |{D|}, |{E|}*{A,B,C},{D},{E}
   c. **Edge *(A,C,3)(A, C, 3)*(A,C,3):**
      i. Skip (forms a cycle with *(A,B)(A, B)*(A,B) and *(B,C)(B, C)*(B,C)).
   d. **Edge *(C,D,4)(C, D, 4)*(C,D,4):**
      i. Add to MST (no cycle).
      ii. Current MST: *(A,B,1),(B,C,2),(C,D,4)(A, B, 1), (B, C, 2), (C, D, 4)*(A,B,1),(B,C,2),(C,D,4)
      iii. Sets: *{A,B,C,D},{E}|{A, B, C, D|}, |{E|}*{A,B,C,D},{E}
   e. **Edge *(D,E,5)(D, E, 5)*(D,E,5):**
      i. Add to MST (no cycle).
      ii. Current MST: *(A,B,1),(B,C,2),(C,D,4),(D,E,5)(A, B, 1), (B, C, 2), (C, D, 4), (D, E, 5)*(A,B,1),(B,C,2),(C,D,4),(D,E,5)
      iii. Sets: *{A,B,C,D,E}|{A, B, C, D, E|}*{A,B,C,D,E}
   f. **Edge *(B,E,6)(B, E, 6)*(B,E,6):**
      i. Skip (forms a cycle with existing MST edges).

3. **Final MST:**
    a. Edges: *(A,B,1),(B,C,2),(C,D,4),(D,E,5)(A, B, 1), (B, C, 2), (C, D, 4), (D, E, 5)*(A,B,1),(B,C,2),(C,D,4),(D,E,5)
    b. Total Weight: *1+2+4+5=121 + 2 + 4 + 5 = 12*1+2+4+5=12

Visual Representation of the MST

Here's how the MST looks:



2. Is the MST Unique?

Conditions for Uniqueness of an MST:

1. If all edge weights are distinct, the MST is guaranteed to be unique.
2. If the graph contains multiple edges with the same weight:
    a. Different MSTs may exist if choosing one edge over another still satisfies the MST conditions.

Is the MST Unique for This Graph?

- **Analyze the Edge Weights**: If there are duplicate weights, check if alternate MSTs are possible.
- If the graph contains distinct edge weights, the MST will be unique.

3. Calculate Shortest Paths from Node A to All Other Nodes (Dijkstra's Algorithm)

What is Dijkstra's Algorithm?

- Dijkstra’s algorithm finds the shortest path from a source node to all other nodes in a weighted graph.

Steps for Dijkstra's Algorithm:

1. **Initialize Distances**:
    a. Set distance to the source node $A=0A = 0$A=0.
    b. Set distance to all other nodes = $\infty|infty$∞.
2. **Priority Queue**:
    a. Use a priority queue to pick the node with the smallest current distance.
3. **Relaxation**:
    a. For each adjacent node, update its distance if a shorter path is found.
4. **Repeat**:
    a. Continue until all nodes are processed.

Example:

- Compute shortest paths step-by-step for the graph, showing updates to the distance table and priority queue.

4. Explain Critical Edges and Identify Them

What is a Critical Edge?

- A critical edge is an edge that, if removed, increases the number of connected components in the graph (i.e., disconnects the graph).

Steps to Find Critical Edges:

1. Remove an edge and check if the graph remains connected.
2. If the graph becomes disconnected, the removed edge is critical.
3. Repeat for all edges.

Example:

- Remove a single edge and explain if the graph remains connected. Show the resulting connected components.

Conclusion:

- If no edge removal increases the number of connected components, the graph has no critical edges.

5. Explain Articulation Points and Identify Them

What is an Articulation Point?

- An articulation point (or cut vertex) is a node that, if removed, increases the number of connected components in the graph.

Steps to Find Articulation Points:

1. Remove a node and all its edges.
2. Check if the graph remains connected.
3. If the graph becomes disconnected, the removed node is an articulation point.
4. Repeat for all nodes.

Example:

- Remove a vertex (e.g., BBB) and analyze connectivity.

Conclusion:

- If no vertex removal increases the number of connected components, the graph has no articulation points.

6. Alternative Path Without Critical Edges or Articulation Points

If the graph has no critical edges or articulation points:

- Removing any single edge or node will not disconnect the graph.
- If you are at BB and CC becomes unavailable, there **must** be another path to EE because the graph remains connected.

Conclusion:

- Without calculations, you can be sure there is another path to EE.

7. Graph Robustness and Its Connection to Critical Edges/Articulation Points

What is Graph Robustness?

- Graph robustness measures how resilient a graph is to failures (e.g., removal of edges or vertices).
- A robust graph remains connected even after failures.

Role of Critical Edges:

- A graph with no critical edges is more robust because no single edge removal can disconnect the graph.

Role of Articulation Points:

- A graph with no articulation points is more robust because no single vertex removal can disconnect the graph.

Key Takeaway:

- To ensure robustness, design graphs with redundancy (multiple paths between nodes) to avoid critical edges and articulation points.