Exercise set 10.5

7. Find all terminal and internal vertices for the following trees:

a. Tree structure:

- **Terminal vertices** (degree = 1): $v_2, v_3, v_4, v_5, v_6, v_7$
- **Internal vertices** (degree > 1): v_1

b. Tree structure:

- **Terminal vertices** (degree = 1): v_1, v_2, v_6, v_7, v_8
- **Internal vertices** (degree > 1): v_3, v_4, v_5

8-13. Graph Creation or Explanation

8. Tree, nine vertices, nine edges

- **Response**: This graph cannot exist because a tree with nine vertices must have exactly eight edges (a tree with n vertices has $n - 1$ edges).

9. Graph, connected, nine vertices, nine edges

- **Example**: A cycle with nine vertices (each vertex connected to two others) plus one additional edge to form a chord, thereby maintaining connectivity and the exact edge count.

10. Graph, circuit-free, nine vertices, six edges

- **Example**: A forest composed of trees; for example, three separate trees within the graph, each a simple path or star configuration, ensuring no circuits are formed.

11. Tree, six vertices, total degree 14

- **Response**: This graph cannot exist because the sum of the degrees of a tree with six vertices must be exactly ten (since a tree with n vertices has a total degree of $2(n - 1)$).

12. Tree, five vertices, total degree 8

- **Example**: A simple star configuration where one central vertex connects to the other four.

13. Graph, connected, six vertices, five edges, nontrivial circuit

- **Example**: A single component graph with five edges forming a cycle (e.g., pentagon) with one vertex connected to the cycle but not part of it, ensuring connectivity and a nontrivial circuit.

Exercise set 10.6

1. Tree Analysis

Given a tree structure with root a:

- **a. Level of** n: Level 3 (three edges from root a to n)
- **b. Level of** a: Level 0 (root level)
- **c. Height of tree**: 3 (maximum level from root to leaf)
- **d. Children of** n: o, u, v, w
- **e. Parent of** g: f
- **f. Siblings of** j: k, l, q, r (all children of same parent e)
- **g. Descendants of** f: g, j, k, l, q, r, s, x (all nodes below f in the tree)

Exercise set 10.7

Using Prim's Algorithm:

1. Initialize a tree with a single vertex (chosen arbitrarily).
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat until all vertices are in the tree.

Dijkstra's Algorithm for Shortest Path

1. Assign to every node a tentative distance value: set it to zero for the initial node and to infinity for all other nodes.
2. Set the initial node as current. Mark all other nodes unvisited. Create a set of all the unvisited nodes called the unvisited set.
3. For the current node, consider all its unvisited neighbors and calculate their tentative distances. Compare the newly calculated tentative distance to the current assigned value and assign the smaller one.
4. When we are done considering all the neighbors of the current node, mark the current node as visited and remove it from the unvisited set. A visited node will not be checked again.
5. If the destination node has been marked visited or if the smallest tentative distance among the nodes in the unvisited set is infinity, then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node," and go back to step 3.

Kruskal's and Prim's Algorithm for Minimum Spanning Tree (MST)

Using Kruskal's Algorithm:

1. Sort all edges in non-decreasing order of their weight.
2. Pick the smallest edge. Check if it forms a cycle with the spanning tree formed so far. If cycle is not formed, include this edge. Else, discard it.
3. Repeat until there are $V - 1$ edges in the spanning tree (V is the number of vertices in the graph).

Using Prim's Algorithm:

1. Initialize a tree with a single vertex (chosen arbitrarily).
2. Grow the tree by one edge: of the edges that connect the tree to vertices not yet in the tree, find the minimum-weight edge, and transfer it to the tree.
3. Repeat until all vertices are in the tree.

Initialization:

1. **Set** all node distances to infinity, except for the starting node a, which is set to 0.
2. **Mark** all nodes as unvisited.

Vertex Setup:

- **a** (distance from a = 0)
- **b, c, d, e, z** (distance from a = infinity initially)

Table Setup:

The table will have columns for:

- **Vertex**
- **Shortest Distance from a** (updated as we find shorter paths)
- **Previous Vertex** (to trace the path)

Table to Track the Steps:

Vertex	Shortest Distance	Previous Vertex
a	0	None
b	∞	-
c	∞	-
d	∞	-
e	∞	-
z	∞	-

Step-by-Step Execution:

1. **Start at a**:
   - Consider neighbors b and d.
   - Update b to 2 (via a), and d to 1 (via a).
   - Update the table.

Vertex	Shortest Distance	Previous Vertex
a	0	None
b	2	a
c	∞	-
d	1	a
e	∞	-
z	∞	-

2. **Move to d** (shortest unvisited vertex):
   - Consider neighbors b, e.
   - Update b to 6 if less, not needed here.
   - Update e to 3 (1 from a to d + 2 from d to e).
   - Update the table.

Vertex	Shortest Distance	Previous Vertex
a	0	None
b	2	a
c	∞	-
d	1	a
e	3	d
z	∞	-

3. **Move to b**:
   - Consider neighbor c.
   - Update c to 5 (2 from a to b + 3 from b to c).
   - Update the table.

Vertex	Shortest Distance	Previous Vertex
a	0	None
b	2	a
c	∞	-
d	1	a
e	3	d
z	∞	-

3. **Move to b**:
   - Consider neighbor c.
   - Update c to 5 (2 from a to b + 3 from b to c).
   - Update the table.

Vertex	Shortest Distance	Previous Vertex
a	0	None
b	2	a
c	5	b
d	1	a
e	3	d
z	∞	-

4. **Move to e**:
   - Consider neighbor z.
   - Update z to 5 (3 from a to e + 2 from e to z).
   - Update the table.

Vertex	Shortest Distance	Previous Vertex
a	0	None
b	2	a
c	5	b
d	1	a
e	3	d
z	5	e

5. **Move to c**:
   - Consider neighbor z.
   - Check if updating z from 13 (5 from a to c + 8 from c to z) improves; it does not here.
   - Finish as all vertices are visited.

The shortest path from a to z is 5, traveling through vertices a to d to e to z.