In [1]:
```
graph_structure = {
    'V1': ['V2', 'V3'],
    'V2': ['V1', 'V3', 'V4'],
    'V3': ['V1', 'V2', 'V5'],
    'V4': ['V2', 'V5'],
    'V5': ['V3', 'V4'],
}

node_list = list(graph_structure.keys())
matrix_size = len(node_list)
node_to_index = {node: index for index, node in enumerate(node_list)}

adjacency_matrix = [[0] * matrix_size for _ in range(matrix_size)]
for node, connected_nodes in graph_structure.items():
    for connected_node in connected_nodes:
        adjacency_matrix[node_to_index[node]][node_to_index[connected_node]] = 1

edge_list = []
for node, connections in graph_structure.items():
    for connection in connections:
        if {node, connection} not in edge_list:
            edge_list.append({node, connection})

total_edges = len(edge_list)
incidence_matrix = [[0] * total_edges for _ in range(matrix_size)]
for edge_index, edge in enumerate(edge_list):
    for node in edge:
        incidence_matrix[node_to_index[node]][edge_index] = 1

node_degrees = [sum(line) for line in adjacency_matrix]
for node, degree in zip(node_list, node_degrees):
    print(f"Node {node} is connected to {degree} edges.")

max_degree_nodes = [node for node, degree in zip(node_list, node_degrees) if degree == max(node_degrees)]
min_degree_nodes = [node for node, degree in zip(node_list, node_degrees) if degree == min(node_degrees)]
print("Maximum connected nodes:", max_degree_nodes)
print("Minimum connected nodes:", min_degree_nodes)

degree_groups = {}
```

```python
for node, degree in zip(node_list, node_degrees):
    if degree not in degree_groups:
        degree_groups[degree] = []
    degree_groups[degree].append(node)

for degree, nodes_with_same_degree in degree_groups.items():
    if len(nodes_with_same_degree) > 1:
        print(f"Nodes with {degree} connections: {nodes_with_same_degree}")

def find_shortest_path(start_node, end_node):
    unvisited = set(node_list)
    shortest_distances = {node: float('inf') for node in node_list}
    shortest_distances[start_node] = 0
    while unvisited:
        current = min(unvisited, key=lambda node: shortest_distances[node])
        unvisited.remove(current)
        for adjacent_index, is_connected in enumerate(adjacency_matrix[node_to_index[current]]):
            if is_connected and node_list[adjacent_index] in unvisited:
                new_distance = shortest_distances[current] + 1
                if new_distance < shortest_distances[node_list[adjacent_index]]:
                    shortest_distances[node_list[adjacent_index]] = new_distance
        if current == end_node:
            break
    return shortest_distances[end_node]

print("Shortest distance from V1 to V5:", find_shortest_path('V1', 'V5'))

def compute_mst():
    edges_in_mst = []
    in_mst = {node: False for node in node_list}
    cheapest_edge_to_node = {node: float('inf') for node in node_list}
    parent_node = {node: None for node in node_list}
    cheapest_edge_to_node[node_list[0]] = 0
    while len(edges_in_mst) < matrix_size - 1:
        next_node = min((node for node in node_list if not in_mst[node]), key=lambda node: cheapest_edge_to_node[node])
        in_mst[next_node] = True
        if parent_node[next_node]:
            edges_in_mst.append((parent_node[next_node], next_node))
        for adjacent_node in node_list:
            if adjacency_matrix[node_to_index[next_node]][node_to_index[adjacent_node]] and not in_mst[adjacent_node] and \
               adjacency_matrix[node_to_index[next_node]][node_to_index[adjacent_node]] < cheapest_edge_to_node[adjacent_node]
```

```
                cheapest_edge_to_node[adjacent_node] = adjacency_matrix[node_to_index[next_node]][node_to_index[adjacent_node]
                parent_node[adjacent_node] = next_node
    return edges_in_mst

print("Edges in the Minimum Spanning Tree:", compute_mst())
```

```
Node V1 is connected to 2 edges.
Node V2 is connected to 3 edges.
Node V3 is connected to 3 edges.
Node V4 is connected to 2 edges.
Node V5 is connected to 2 edges.
Maximum connected nodes: ['V2', 'V3']
Minimum connected nodes: ['V1', 'V4', 'V5']
Nodes with 2 connections: ['V1', 'V4', 'V5']
Nodes with 3 connections: ['V2', 'V3']
Shortest distance from V1 to V5: 2
Edges in the Minimum Spanning Tree: [('V1', 'V2'), ('V1', 'V3'), ('V2', 'V4'), ('V3', 'V5')]
```