Lab Assignment 8 - Graph Matrices and Operations

This notebook addresses the tasks given in Lab Assignment 8 for the course Foundation of Data Science.

Tasks

1. Write Python code to create the incidence and adjacency matrices for a given graph.
2. Display the total number of edges.
3. Display the number of edges connected with each node.
4. Identify nodes with maximum and minimum edges.
5. Find nodes with equal number of edges.
6. Calculate and display the shortest path between nodes.
7. Compute and display the minimum spanning tree.

First, we will define the graph for which we need to perform these operations.

In [22]:
```
graph = {
    'A': ['B', 'C'],
    'B': ['A', 'C', 'D'],
    'C': ['A', 'B', 'E'],
    'D': ['B', 'E'],
    'E': ['C', 'D'],
}
```

In [23]:
```
def adjacency_matrix(graph):
    nodes = list(graph.keys())
    size = len(nodes)
    matrix = [[0]*size for _ in range(size)]
    node_index = {node: idx for idx, node in enumerate(nodes)}

    for node, adjacents in graph.items():
        for adjacent in adjacents:
```

```python
            matrix[node_index[node]][node_index[adjacent]] = 1
    return matrix
```

In [24]:
```python
def incidence_matrix(graph):
    nodes = list(graph.keys())
    edges = set()
    for node, adjacents in graph.items():
        for adjacent in adjacents:
            edges.add(tuple(sorted([node, adjacent])))
    edges = list(edges)
    size_nodes = len(nodes)
    size_edges = len(edges)
    matrix = [[0]*size_edges for _ in range(size_nodes)]
    node_index = {node: idx for idx, node in enumerate(nodes)}
    edge_index = {edge: idx for idx, edge in enumerate(edges)}

    for node, adjacents in graph.items():
        for adjacent in adjacents:
            edge = tuple(sorted([node, adjacent]))
            matrix[node_index[node]][edge_index[edge]] = 1
    return matrix
```

In [25]:
```python
adj_matrix = adjacency_matrix(graph)
inc_matrix = incidence_matrix(graph)
```

In [26]:
```python
total_edges = len(inc_matrix[0])
```

In [27]:
```python
print("Adjacency Matrix:")
for row in adj_matrix:
    print(row)
print("\nIncidence Matrix:")
for row in inc_matrix:
    print(row)
print("\nTotal number of edges:", total_edges)
```

```
Adjacency Matrix:
[0, 1, 1, 0, 0]
[1, 0, 1, 1, 0]
[1, 1, 0, 0, 1]
[0, 1, 0, 0, 1]
[0, 0, 1, 1, 0]

Incidence Matrix:
[1, 1, 0, 0, 0, 0]
[1, 0, 1, 0, 0, 1]
[0, 1, 0, 1, 0, 1]
[0, 0, 1, 0, 1, 0]
[0, 0, 0, 1, 1, 0]

Total number of edges: 6
```

In [28]:

```python
degrees = [sum(row) for row in adj_matrix]
nodes = list(graph.keys())
node_index = {node: idx for idx, node in enumerate(nodes)}
size = len(nodes)

for node, degree in zip(nodes, degrees):
    print(f"Node {node} has {degree} edges.")
```

```
Node A has 2 edges.
Node B has 3 edges.
Node C has 3 edges.
Node D has 2 edges.
Node E has 2 edges.
```

In [29]:

```python
max_degree = max(degrees)
min_degree = min(degrees)
max_nodes = [node for node, degree in zip(nodes, degrees) if degree == max_degree]
min_nodes = [node for node, degree in zip(nodes, degrees) if degree == min_degree]

print("Nodes with maximum edges:", max_nodes)
print("Nodes with minimum edges:", min_nodes)
```

```
Nodes with maximum edges: ['B', 'C']
Nodes with minimum edges: ['A', 'D', 'E']
```

In [30]:
```
equal_edge_nodes = {}
for node, degree in zip(nodes, degrees):
    if degree not in equal_edge_nodes:
        equal_edge_nodes[degree] = []
    equal_edge_nodes[degree].append(node)

for degree, node_list in equal_edge_nodes.items():
    if len(node_list) > 1:
        print(f"Nodes with {degree} edges: {node_list}")
```

```
Nodes with 2 edges: ['A', 'D', 'E']
Nodes with 3 edges: ['B', 'C']
```

In [31]:
```
def dijkstra(start, target):
    unvisited = set(nodes)
    shortest_path = {node: float('inf') for node in nodes}
    shortest_path[start] = 0
    current_node = start
    while unvisited:
        current_node = min(unvisited, key=lambda node: shortest_path[node])
        unvisited.remove(current_node)
        for idx, connected in enumerate(adj_matrix[node_index[current_node]]):
            if connected and nodes[idx] in unvisited:
                distance = shortest_path[current_node] + 1
                if distance < shortest_path[nodes[idx]]:
                    shortest_path[nodes[idx]] = distance
        if current_node == target:
            break
    return shortest_path[target]
```

In [32]:
```
print("Shortest path from A to E:", dijkstra('A', 'E'))
```

```
Shortest path from A to E: 2
```

In [33]:
```
def prim():
    in_tree = {node: False for node in nodes}
    min_edge = {node: float('inf') for node in nodes}
    parent = {node: None for node in nodes}
    min_edge[nodes[0]] = 0
    mst_edges = []
```

```
    while len(mst_edges) < size - 1:
        u = min((node for node in nodes if not in_tree[node]), key=lambda node: min_edge[node])
        in_tree[u] = True
        if parent[u]:
            mst_edges.append((parent[u], u))

        for v in nodes:
            if adj_matrix[node_index[u]][node_index[v]] and not in_tree[v] and adj_matrix[node_index[u]][node_index[v]] < min_
                min_edge[v] = adj_matrix[node_index[u]][node_index[v]]
                parent[v] = u

    return mst_edges
```

In [34]:
```
print("Minimum spanning tree:", prim())
```

```
Minimum spanning tree: [('A', 'B'), ('A', 'C'), ('B', 'D'), ('C', 'E')]
```