Create an empty graph G.

Add the following nodes: A, B, C, D

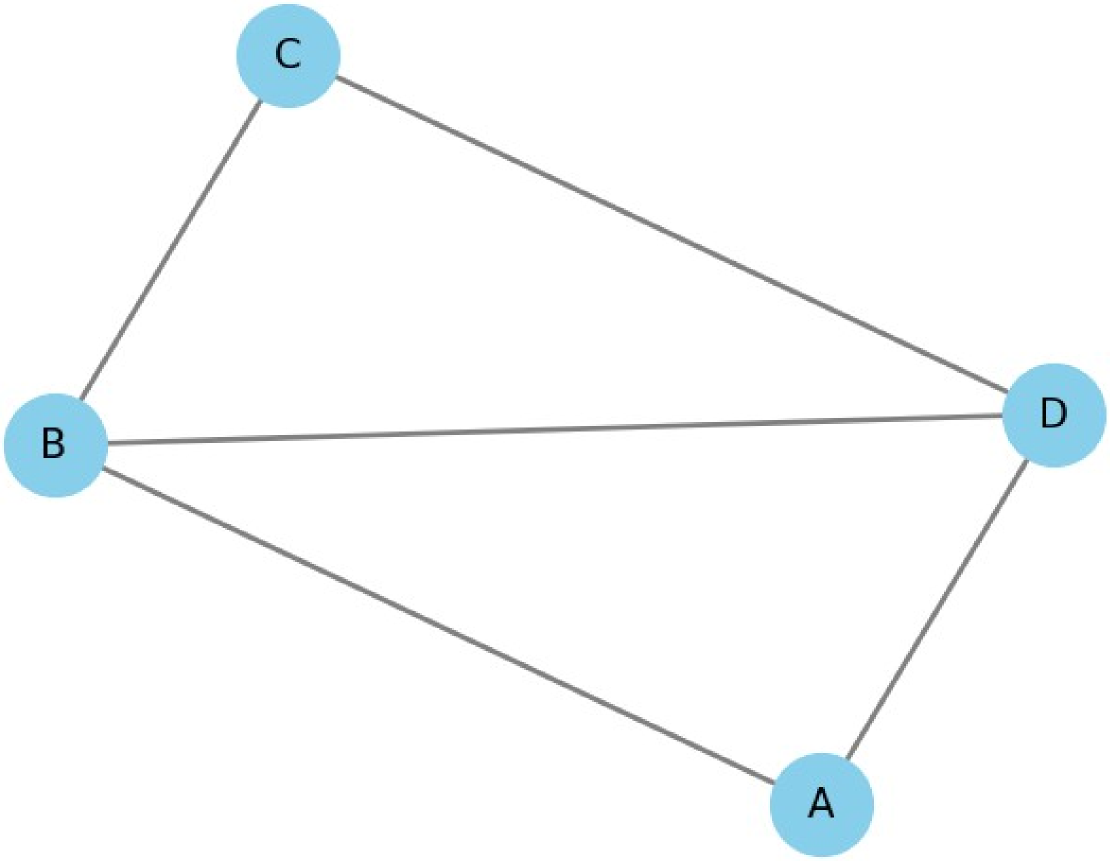
Add the following edges: A-B, B-C, C-D, D-A, B-D

```
import networkx as nx
# Create an empty graph G.
G = nx.Graph()

# Add the following nodes: A, B, C, D
nodes = ['A', 'B', 'C', 'D']
G.add_nodes_from(nodes)

# Add the following edges: A-B, B-C, C-D, D-A, B-D
edges = [('A', 'B'), ('B', 'C'), ('C', 'D'), ('D', 'A'), ('B', 'D')]
G.add_edges_from(edges)

# Print the nodes and edges:
print("Nodes:", G.nodes())
print("Edges:", G.edges())
```

```
Nodes: ['A', 'B', 'C', 'D']
Edges: [('A', 'B'), ('A', 'D'), ('B', 'C'), ('B', 'D'), ('C', 'D')]
```

Visualize the Graph

```
#Visualize the Graph
import matplotlib.pyplot as plt
nx.draw(G, with_labels=True, node_color='skyblue', node_size=1500,
edge_color='gray', width=2, font_size=15)
plt.show()
```

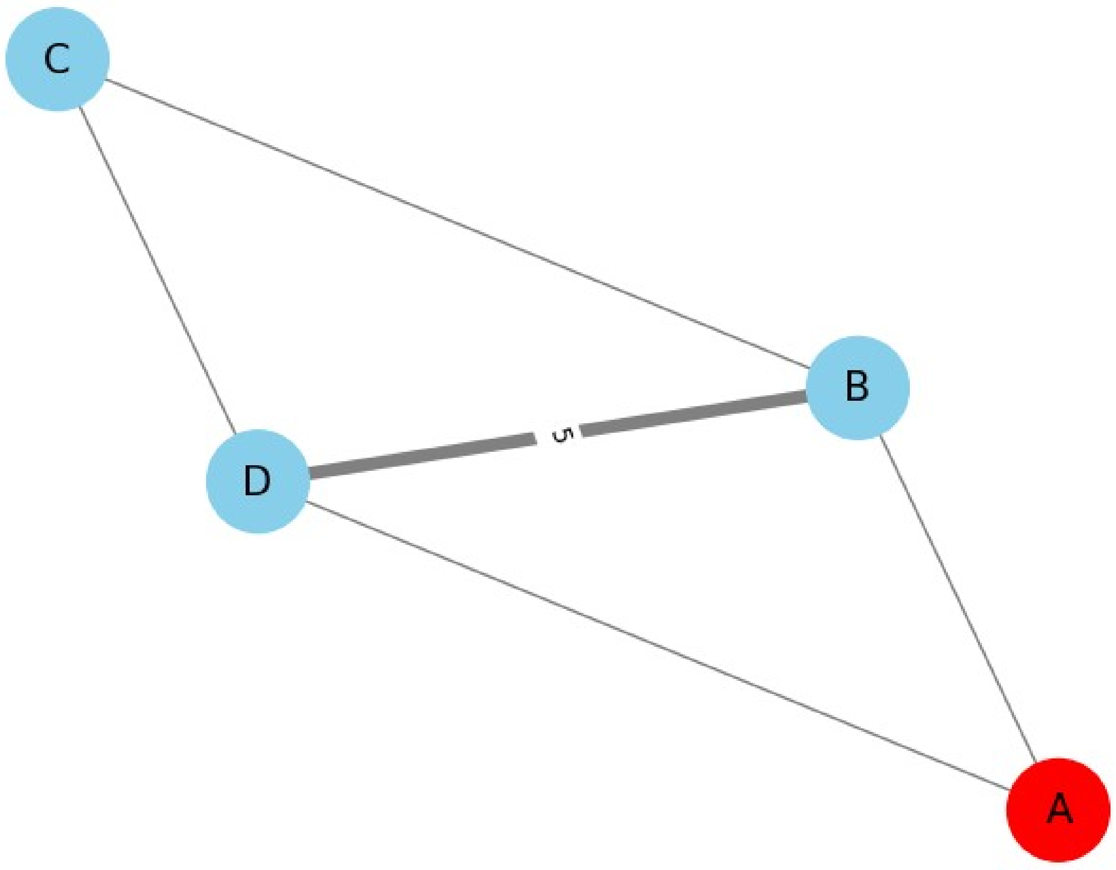


Add an attribute color to node A with value 'red'.

Add a weight attribute to edge (B, D) with value 5.

```python
# Adding an attribute color to node A with value 'red'.
G.nodes['A']['color'] = 'red'

# Adding a weight attribute to edge (B, D) with value 5.
G.edges['B', 'D']['weight'] = 5

# Visualize the graph with node colors and edge weights
node_colors = [G.nodes[node].get('color', 'skyblue') for node in
G.nodes()]
edge_weights = [G.edges[u, v].get('weight', 1) for u, v in G.edges()]

nx.draw(G, with_labels=True, node_color=node_colors, node_size=1500,
        edge_color='gray', width=edge_weights, font_size=15)

# Add labels for edge weights
edge_labels = nx.get_edge_attributes(G, 'weight')
nx.draw_networkx_edge_labels(G, pos=nx.spring_layout(G),
edge_labels=edge_labels)
```

```
plt.show()

print("Nodes with attributes:", G.nodes(data=True))
print("Edges with attributes:", G.edges(data=True))
```



```
Nodes with attributes: [('A', {'color': 'red'}), ('B', {}), ('C', {}),
('D', {})]
Edges with attributes: [('A', 'B', {}), ('A', 'D', {}), ('B', 'C',
{}), ('B', 'D', {'weight': 5}), ('C', 'D', {})]
```

How many nodes are there in the graph?

How many edges are there?

List all the neighbors of node B.

```
# No.of nodes in the graph
num_nodes = G.number_of_nodes()
print(f"Number of nodes: {num_nodes}")
```

```
# No.of edges in the graph
num_edges = G.number_of_edges()
print(f"Number of edges: {num_edges}")

# Listing all the neighbors of node B.
neighbors_B = list(G.neighbors('B'))
print(f"Neighbors of node B: {neighbors_B}")
```

```
Number of nodes: 4
Number of edges: 5
Neighbors of node B: ['A', 'C', 'D']
```

Find the shortest path between node A and node C (assuming unweighted edges).

```
# Finding the shortest path between node A and node C (assuming
unweighted edges).

shortest_path = nx.shortest_path(G, source='A', target='C')
print(f"Shortest path between A and C: {shortest_path}")
```

```
Shortest path between A and C: ['A', 'B', 'C']
```