15+1 Bonus

Q1_Solution

January 22, 2026

1 Assignment 1 - Question 1

1.1 Modeling a Graph from My Life

Good!

1.1.1 Graph Description:

- **Vertices**: Represent different locations/places in my daily life
    - Home: My home
    - School: School
    - Library: Library
    - Gym: Gymnasium
    - Mall: Shopping mall
    - Park: Park
    - Restaurant: Restaurant
    - Friend1: Friend 1's home
    - Friend2: Friend 2's home
    - Work: Workplace
    - Cinema: Movie theater
    - Coffee: Coffee shop
- **Edges**: Represent transportation routes between locations
    - "B" = Bus route
    - "W" = Walking route
    - "M" = Metro route
    - "C" = Car route

```python
[1]: from IPython.core.interactiveshell import InteractiveShell
     InteractiveShell.ast_node_interactivity = 'all'
```

```python
[2]: # Creating my graph with at least 10 vertices

     G = Graph({
         "Home": {"School": "B", "Gym": "W", "Library": "B", "Friend1": "M",
     ↪"Coffee": "W"},
         "School": {"Library": "W", "Gym": "B", "Friend1": "B", "Work": "M"},
         "Library": {"Park": "W", "Coffee": "B", "Mall": "B"},
         "Gym": {"Park": "W", "Restaurant": "B", "Friend2": "W"},
         "Mall": {"Cinema": "W", "Restaurant": "W", "Friend2": "B"},
         "Park": {"Restaurant": "W", "Coffee": "W"},
         "Restaurant": {"Cinema": "W", "Friend1": "B"},
```

```
        "Friend1": {"Friend2": "M", "Work": "B"},
        "Friend2": {"Work": "B", "Cinema": "B"},
        "Work": {"Coffee": "M", "Cinema": "B"},
        "Cinema": {"Coffee": "W"},
        "Coffee": {"Home": "W", "Library": "B", "Park": "W", "Work": "M", "Cinema":␣
    ↪"W"}
})

print(f"Number of vertices: {G.order()}")
print(f"Number of edges: {G.size()}")
print(f"Vertices: {G.vertices()}")
print(f"Edges: {G.edges()}")
```

```
Number of vertices: 12
Number of edges: 29
Vertices: ['Home', 'School', 'Library', 'Gym', 'Mall', 'Park', 'Restaurant',
'Friend1', 'Friend2', 'Work', 'Cinema', 'Coffee']
Edges: [('Home', 'School', 'B'), ('Home', 'Library', 'B'), ('Gym', 'Home', 'W'),
('Friend1', 'Home', 'M'), ('Coffee', 'Home', 'W'), ('Library', 'School', 'W'),
('Gym', 'School', 'B'), ('Friend1', 'School', 'B'), ('School', 'Work', 'M'),
('Library', 'Mall', 'B'), ('Library', 'Park', 'W'), ('Coffee', 'Library', 'B'),
('Gym', 'Park', 'W'), ('Gym', 'Restaurant', 'B'), ('Friend2', 'Gym', 'W'),
('Mall', 'Restaurant', 'W'), ('Friend2', 'Mall', 'B'), ('Cinema', 'Mall', 'W'),
('Park', 'Restaurant', 'W'), ('Coffee', 'Park', 'W'), ('Friend1', 'Restaurant',
'B'), ('Cinema', 'Restaurant', 'W'), ('Friend1', 'Friend2', 'M'), ('Friend1',
'Work', 'B'), ('Friend2', 'Work', 'B'), ('Cinema', 'Friend2', 'B'), ('Cinema',
'Work', 'B'), ('Coffee', 'Work', 'M'), ('Cinema', 'Coffee', 'W')]
```
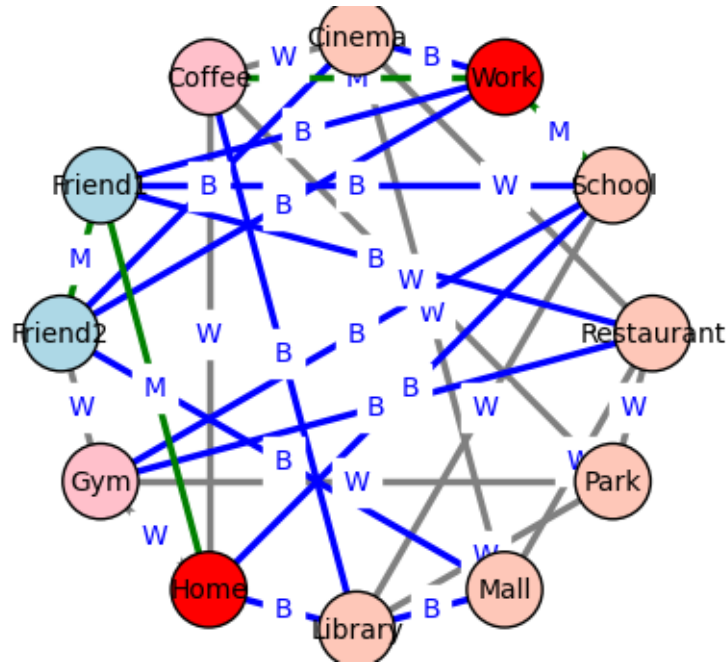
[3]:
```
# Plotting my graph with styling

G.show(
    vertex_colors={"red": ["Home", "Work"], "pink": ["Gym", "Coffee"],␣
    ↪"lightblue": ["Friend1", "Friend2"]},
    edge_thickness=2.5,
    edge_colors={
        "gray": [(a, b) for a, b, c in G.edges() if G.edge_label(a, b) == "W"],
        "blue": [(a, b) for a, b, c in G.edges() if G.edge_label(a, b) == "B"],
        "green": [(a, b) for a, b, c in G.edges() if G.edge_label(a, b) == "M"]
    },
    edge_labels=True,
    vertex_size=800,
    layout="circular"
)
```

[3]:
```

## 1.2 Required Questions:

### 1.2.1 1. Is my graph connected?

```
[4]: # Question 1: Is my graph connected?
     is_connected = G.is_connected()
     print(f"Is my graph connected? {is_connected}")

     if not is_connected:
         print(f"The graph has {len(G.connected_components())} connected components:
     ↪")
         for i, component in enumerate(G.connected_components(), 1):
             print(f"  Component {i}: {component}")
```

Is my graph connected? True

1.2.2 2. Size of the largest group of vertices that are mutually adjacent (clique number)?

```
[5]: # Question 2: Clique number
     clique_num = G.clique_number()
     print(f"The size of the largest clique: {clique_num}")
```

The size of the largest clique: 3

1.2.3 3. Present such cliques (maximum cliques)

```
[6]: # Question 3: Maximum cliques
     max_cliques = G.cliques_maximum()
     print(f"Maximum cliques: {max_cliques}")
     print(f"Number of maximum cliques: {len(max_cliques)}")

     for i, clique in enumerate(max_cliques, 1):
         print(f"  Clique {i}: {clique}")
```

```
Maximum cliques: [['Friend1', 'Friend2', 'Work'], ['Friend2', 'Work', 'Cinema'],
['Gym', 'Park', 'Restaurant'], ['Home', 'Library', 'Coffee'], ['Home', 'School',
'Friend1'], ['Home', 'School', 'Gym'], ['Home', 'School', 'Library'],
['Library', 'Park', 'Coffee'], ['Mall', 'Friend2', 'Cinema'], ['Mall',
'Restaurant', 'Cinema'], ['School', 'Friend1', 'Work'], ['Work', 'Cinema',
'Coffee']]
Number of maximum cliques: 12
  Clique 1: ['Friend1', 'Friend2', 'Work']
  Clique 2: ['Friend2', 'Work', 'Cinema']
  Clique 3: ['Gym', 'Park', 'Restaurant']
  Clique 4: ['Home', 'Library', 'Coffee']
  Clique 5: ['Home', 'School', 'Friend1']
  Clique 6: ['Home', 'School', 'Gym']
  Clique 7: ['Home', 'School', 'Library']
  Clique 8: ['Library', 'Park', 'Coffee']
  Clique 9: ['Mall', 'Friend2', 'Cinema']
  Clique 10: ['Mall', 'Restaurant', 'Cinema']
  Clique 11: ['School', 'Friend1', 'Work']
  Clique 12: ['Work', 'Cinema', 'Coffee']
```

1.2.4 4. How many vertices to remove in order to get a disconnected graph? (Vertex connectivity)

```
[7]: # Question 4: Vertex connectivity
     vertex_conn = G.vertex_connectivity()
     print(f"Vertex connectivity: {vertex_conn}")
     print(f"This means we need to remove at least {vertex_conn} vertices to␣
      ↪disconnect the graph.")

     if vertex_conn > 0:
         min_cut = G.vertex_connectivity(value_only=False)
         print(f"A minimum vertex cut: {min_cut}")
```

```
Vertex connectivity: 4
This means we need to remove at least 4 vertices to disconnect the graph.
A minimum vertex cut: (4.0, ['Library', 'Restaurant', 'Friend2', 'Cinema'])
```

1.3 Additional Meaningful Questions:

1.3.1 5. What is the degree of each vertex?

```python
# Additional Question 1: Degree of each vertex
print("Degree of each vertex:")
for vertex in sorted(G.vertices()):
    degree = G.degree(vertex)
    neighbors = list(G.neighbors(vertex))
    print(f"  {vertex}: degree = {degree}, neighbors = {neighbors}")
```

```
Degree of each vertex:
  Cinema: degree = 5, neighbors = ['Mall', 'Restaurant', 'Friend2', 'Work',
'Coffee']
  Coffee: degree = 5, neighbors = ['Home', 'Library', 'Park', 'Work', 'Cinema']
  Friend1: degree = 5, neighbors = ['Home', 'School', 'Restaurant', 'Friend2',
'Work']
  Friend2: degree = 5, neighbors = ['Gym', 'Mall', 'Friend1', 'Work', 'Cinema']
  Gym: degree = 5, neighbors = ['Home', 'School', 'Park', 'Restaurant',
'Friend2']
  Home: degree = 5, neighbors = ['School', 'Library', 'Gym', 'Friend1',
'Coffee']
  Library: degree = 5, neighbors = ['Home', 'School', 'Mall', 'Park', 'Coffee']
  Mall: degree = 4, neighbors = ['Library', 'Restaurant', 'Friend2', 'Cinema']
  Park: degree = 4, neighbors = ['Library', 'Gym', 'Restaurant', 'Coffee']
  Restaurant: degree = 5, neighbors = ['Gym', 'Mall', 'Park', 'Friend1',
'Cinema']
  School: degree = 5, neighbors = ['Home', 'Library', 'Gym', 'Friend1', 'Work']
  Work: degree = 5, neighbors = ['School', 'Friend1', 'Friend2', 'Cinema',
'Coffee']
```

1.3.2 6. What is the diameter and radius of the graph?

```python
# Additional Question 2: Diameter and radius
if G.is_connected():
    diameter = G.diameter()
    radius = G.radius()
    print(f"Diameter: {diameter}")
    print(f"Radius: {radius}")

    center = G.center()
    print(f"Center vertices: {center}")
else:
    print("Graph is not connected, cannot compute diameter and radius.")
```

```
Diameter: 2
Radius: 2
Center vertices: ['Home', 'School', 'Library', 'Gym', 'Mall', 'Park',
'Restaurant', 'Friend1', 'Friend2', 'Work', 'Cinema', 'Coffee']
```

1.3.3 7. Can I visit all locations using only bus routes?

```python
# Additional Question 3: Bus-only subgraph
BusSubgraph = Graph([(a, b, c) for a, b, c in G.edges() if G.edge_label(a, b)
 == "B"])

print("Bus-only subgraph:")
print(f"  Vertices: {BusSubgraph.vertices()}")
print(f"  Is connected? {BusSubgraph.is_connected()}")
print(f"  Number of vertices: {BusSubgraph.order()}")
print(f"  Number of edges: {BusSubgraph.size()}")

if BusSubgraph.is_connected():
    print("\nYes! I can visit all locations using only bus routes.")
else:
    print("\nNo, I cannot visit all locations using only bus routes.")
    print(f"The bus subgraph has {len(BusSubgraph.connected_components())}
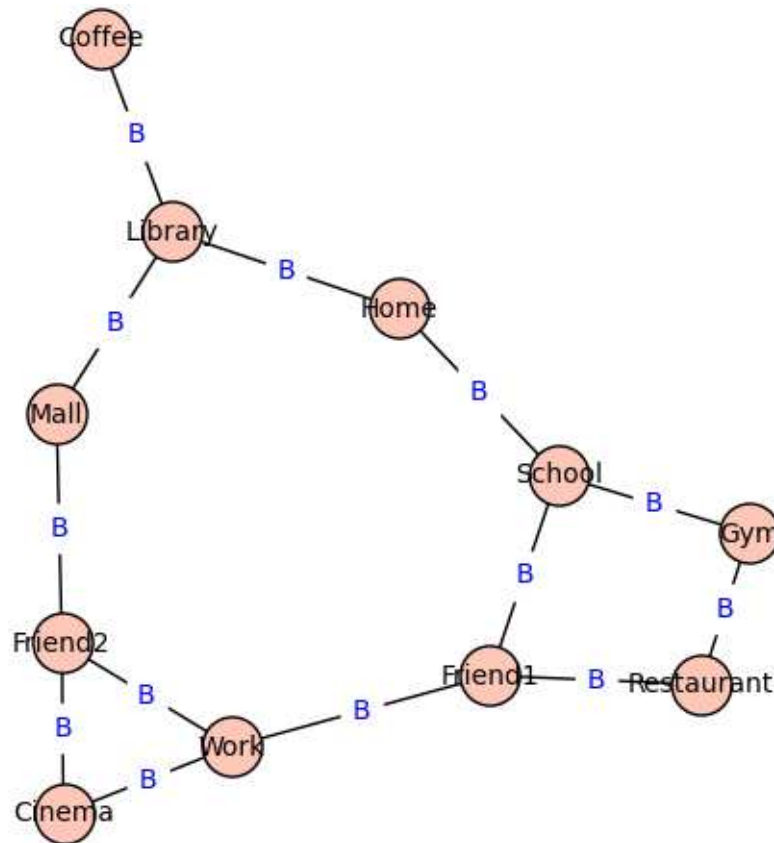 connected components.")

plot(BusSubgraph, vertex_size=500, edge_labels=True, figsize=6)
```

```
Bus-only subgraph:
  Vertices: ['Home', 'School', 'Library', 'Gym', 'Friend1', 'Mall', 'Coffee',
'Restaurant', 'Friend2', 'Work', 'Cinema']
  Is connected? True
  Number of vertices: 11
  Number of edges: 13

Yes! I can visit all locations using only bus routes.
```

[10]:

1.3.4 8. What is the shortest path from Home to each location?

```
[11]: # Additional Question 4: Shortest paths from Home
      if G.is_connected():
          start = "Home"
          print(f"Shortest paths from {start} to all other locations:")

          for target in sorted(G.vertices()):
              if target != start:
                  try:
                      path = G.shortest_path(start, target)
                      length = len(path) - 1
                      print(f"  {start} -> {target}: {path} (length: {length})")
                  except:
                      print(f"  {start} -> {target}: No path exists")
      else:
          print("Graph is not connected, cannot compute shortest paths.")
```

Shortest paths from Home to all other locations:
  Home -> Cinema: ['Home', 'Coffee', 'Cinema'] (length: 2)
  Home -> Coffee: ['Home', 'Coffee'] (length: 1)

```
Home -> Friend1: ['Home', 'Friend1'] (length: 1)
Home -> Friend2: ['Home', 'Gym', 'Friend2'] (length: 2)
Home -> Gym: ['Home', 'Gym'] (length: 1)
Home -> Library: ['Home', 'Library'] (length: 1)
Home -> Mall: ['Home', 'Library', 'Mall'] (length: 2)
Home -> Park: ['Home', 'Library', 'Park'] (length: 2)
Home -> Restaurant: ['Home', 'Gym', 'Restaurant'] (length: 2)
Home -> School: ['Home', 'School'] (length: 1)
Home -> Work: ['Home', 'School', 'Work'] (length: 2)
```

1.3.5   9. Does the graph have a Hamiltonian cycle?

```
[12]: # Additional Question 5: Hamiltonian cycle
if G.is_connected():
    try:
        hamiltonian = G.hamiltonian_cycle()
        print(f"Yes! The graph has a Hamiltonian cycle:")
        print(f"  {hamiltonian}")
        print("\nThis means I can visit all locations exactly once and return␣
 ↪to the starting point.")
    except:
        print("No, the graph does not have a Hamiltonian cycle.")
        print("This means I cannot visit all locations exactly once and return␣
 ↪to the starting point.")
else:
    print("Graph is not connected, cannot have a Hamiltonian cycle.")
```

```
Yes! The graph has a Hamiltonian cycle:
  TSP from

This means I can visit all locations exactly once and return to the starting
point.
```

1.3.6   10. What is the minimum and maximum degree?

```
[13]: # Additional Question 6: Minimum and maximum degree
degrees = [G.degree(v) for v in G.vertices()]
min_degree = min(degrees)
max_degree = max(degrees)
avg_degree = sum(degrees) / len(degrees) if degrees else 0

print(f"Minimum degree: {min_degree}")
print(f"Maximum degree: {max_degree}")
print(f"Average degree: {avg_degree:.2f}")

min_degree_vertices = [v for v in G.vertices() if G.degree(v) == min_degree]
max_degree_vertices = [v for v in G.vertices() if G.degree(v) == max_degree]
```

```python
print(f"\nVertices with minimum degree: {min_degree_vertices}")
print(f"Vertices with maximum degree: {max_degree_vertices}")
```

```
Minimum degree: 4
Maximum degree: 5
Average degree: 4.83

Vertices with minimum degree: ['Mall', 'Park']
Vertices with maximum degree: ['Home', 'School', 'Library', 'Gym', 'Restaurant',
'Friend1', 'Friend2', 'Work', 'Cinema', 'Coffee']
```