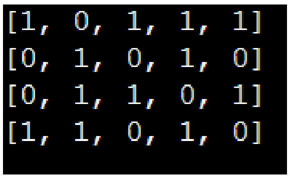
Advanced Data Structures with Python Laboratory (MCAC294)

Assignment – 5

1. Write a Python program to store the following Graph using Adjacency Matrix & display that.

```
class GraphMatrix:
    def __init__(self, vertices):
        self.vertices = vertices
        self.matrix = [[0] * vertices for _ in range(vertices)]
    def add_edge(self, u, v):
        self.matrix[u][v] = 1
        self.matrix[v][u] = 1
    def display(self):
        print("Adjacency Matrix:")
        for row in self.matrix:
            print(row)
g = GraphMatrix(5)
edges = [(0, 1), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (3, 4)]
for u, v in edges:
    g.add_edge(u, v)
g.display()
```

Output:

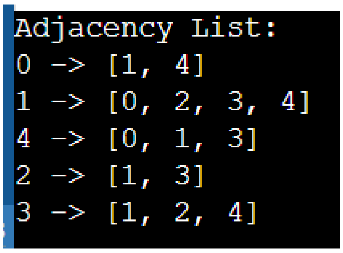


```
[1, 0, 1, 1, 1]
[0, 1, 0, 1, 0]
[0, 1, 1, 0, 1]
[1, 1, 0, 1, 0]
```

2. Write a Python program to store the following Graph using Adjacency List & display that.

```
class GraphList:
    def __init__(self):
        self.graph = {}
    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)
        self.graph[v].append(u)
    def display(self):
        print("Adjacency List:")
        for key, value in self.graph.items():
            print(f"{key} -> {value}")
g = GraphList()
edges = [(0, 1), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (3, 4)]
for u, v in edges:
    g.add_edge(u, v)
g.display()
```

Output:

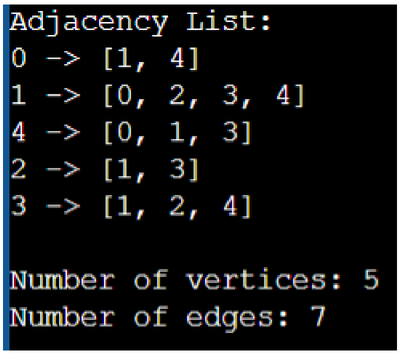


```
Adjacency List:
0 -> [1, 4]
1 -> [0, 2, 3, 4]
4 -> [0, 1, 3]
2 -> [1, 3]
3 -> [1, 2, 4]
```

3. Write a Python program to count number of vertices and edges present in a graph.

```
class GraphList:
    def __init__(self):
        self.graph = {}
    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)
        self.graph[v].append(u)
    def display(self):
        print("Adjacency List:")
        for key, value in self.graph.items():
            print(f"{key} -> {value}")
g = GraphList()
edges = [(0, 1), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (3, 4)]
for u, v in edges:
    g.add_edge(u, v)
g.display()
def count_vertices_edges(graph):
    vertices = len(graph)
    edges = sum(len(neighbors) for neighbors in graph.values()) // 2
    return vertices, edges
vertices, edges = count_vertices_edges(g.graph)
print(f"\nNumber of vertices: {vertices}")
print(f"Number of edges: {edges}")
```

Output:

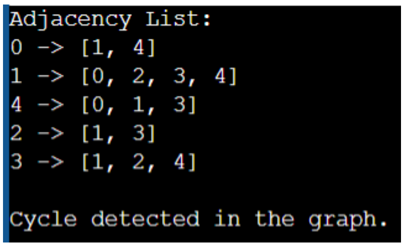


```
Adjacency List:
0 -> [1, 4]
1 -> [0, 2, 3, 4]
4 -> [0, 1, 3]
2 -> [1, 3]
3 -> [1, 2, 4]

Number of vertices: 5
Number of edges: 7
```

4. Write a Python program to detect a cycle in a graph.

```
class GraphList:
    def __init__(self):
        self.graph = {}
    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)
        self.graph[v].append(u)
    def display(self):
        print("Adjacency List:")
        for key, value in self.graph.items():
            print(f"{key} -> {value}")
g = GraphList()
edges = [(0, 1), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (3, 4)]
for u, v in edges:
    g.add_edge(u, v)
g.display()
def detect_cycle(graph):
    visited = set()
    def dfs(node, parent):
        visited.add(node)
        for neighbor in graph[node]:
            if neighbor not in visited:
                if dfs(neighbor, node):
                    return True
            elif parent != neighbor:
                return True
        return False
    for node in graph:
        if node not in visited:
            if dfs(node, -1):
                return True
    return False
if detect_cycle(g.graph):
    print("\nCycle detected in the graph.")
else:
    print("\nNo cycle detected.")
```

Output:

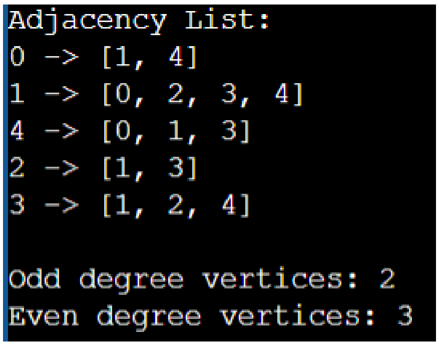


```
Adjacency List:
0 -> [1, 4]
1 -> [0, 2, 3, 4]
4 -> [0, 1, 3]
2 -> [1, 3]
3 -> [1, 2, 4]

Cycle detected in the graph.
```

5. Write a Python program to identify number of odd degree vertices and number of even degree vertices in a graph.

```
class GraphList:
    def __init__(self):
        self.graph = {}
    def add_edge(self, u, v):
        if u not in self.graph:
            self.graph[u] = []
        if v not in self.graph:
            self.graph[v] = []
        self.graph[u].append(v)
        self.graph[v].append(u)
    def display(self):
        print("Adjacency List:")
        for key, value in self.graph.items():
            print(f"{key} -> {value}")
g = GraphList()
edges = [(0, 1), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (3, 4)]
for u, v in edges:
    g.add_edge(u, v)
g.display()
def count_odd_even_vertices(graph):
    odd_count = even_count = 0
    for node in graph:
        degree = len(graph[node])
        if degree % 2 == 0:
            even_count += 1
        else:
            odd_count += 1
    return odd_count, even_count
odd, even = count_odd_even_vertices(g.graph)
print(f"\nOdd degree vertices: {odd}")
print(f"Even degree vertices: {even}")
```

Output:



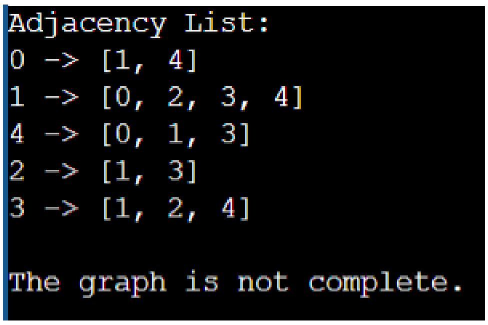
```
Adjacency List:
0 -> [1, 4]
1 -> [0, 2, 3, 4]
4 -> [0, 1, 3]
2 -> [1, 3]
3 -> [1, 2, 4]

Odd degree vertices: 2
Even degree vertices: 3
```

6. Write a Python program to check whether a given graph is complete or not.

```
class GraphList:
   def __init__(self):
      self.graph = {}
   def add_edge(self, u, v):
      if u not in self.graph:
         self.graph[u] = []
      if v not in self.graph:
         self.graph[v] = []
      self.graph[u].append(v)
      self.graph[v].append(u)
   def display(self):
      print("Adjacency List:")
      for key, value in self.graph.items():
         print(f"{key} -> {value}")
g = GraphList()
edges = [(0, 1), (0, 4), (1, 2), (1, 3), (1, 4), (2, 3), (3, 4)]
for u, v in edges:
   g.add_edge(u, v)
g.display()
def is_complete(graph):
   vertices = len(graph)
   expected_edges = vertices * (vertices - 1) // 2
   actual_edges = sum(len(neighbors) for neighbors in
graph.values()) // 2
   return expected_edges == actual_edges
if is_complete(g.graph):
   print("\nThe graph is complete.")
```

```
else:
    print("\nThe graph is not complete.")
```

Output:



```
Adjacency List:
0 -> [1, 4]
1 -> [0, 2, 3, 4]
4 -> [0, 1, 3]
2 -> [1, 3]
3 -> [1, 2, 4]

The graph is not complete.
```