**GRAPH:**

**ADJ LIST:**

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):

for i in self.graph:

print(f"{i} -> {self.graph[i]}")

g = GraphList()

g.add\_edge(1, 2)

g.add\_edge(1, 3)

g.add\_edge(1, 4)

g.add\_edge(2, 4)

g.add\_edge(2, 5)

g.add\_edge(3, 4)

g.add\_edge(3, 5)

g.add\_edge(4, 6)

g.add\_edge(5, 6)

g.display()

**OUTPUT:**

1 -> [2, 3, 4]

2 -> [1, 4, 5]

3 -> [1, 4, 5]

4 -> [1, 2, 3, 6]

5 -> [2, 3, 6]

6 -> [4, 5]

=== Code Execution Successful ===

**class GraphAdjMatrix:**

def \_\_init\_\_(self, size=None, adj\_matrix=None):

if adj\_matrix is not None:

self.adj\_matrix = adj\_matrix

self.size = len(adj\_matrix)

else:

self.size = size

self.adj\_matrix = [[0 for \_ in range(size)] for \_ in range(size)]

def add\_edge(self, u, v):

if u < self.size and v < self.size:

self.adj\_matrix[u][v] = 1

self.adj\_matrix[v][u] = 1

def display(self):

print("Adjacency Matrix:")

for row in self.adj\_matrix:

print(row)

adj\_matrix = [

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

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

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

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

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

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

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

]

g = GraphAdjMatrix(adj\_matrix=adj\_matrix)

g.display()

**OUTPUT:**

Adjacency Matrix:

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

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

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

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

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

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

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

=== Code Execution Successful ===

**ADJ MATRIX-2:**

class GraphMatrix:

def \_\_init\_\_(self, vertices):

self.vertices = vertices

self.graph = [[0 for \_ in range(vertices)] for \_ in range(vertices)]

def add\_edge(self, u, v):

self.graph[u - 1][v - 1] = 1

self.graph[v - 1][u - 1] = 1

def display(self):

print("Adjacency Matrix:")

for row in self.graph:

print(row)

g = GraphMatrix(6)

g.add\_edge(1, 2)

g.add\_edge(1, 3)

g.add\_edge(1, 4)

g.add\_edge(2, 4)

g.add\_edge(2, 5)

g.add\_edge(3, 4)

g.add\_edge(3, 5)

g.add\_edge(4, 6)

g.add\_edge(5, 6)

g.display()

**OUTPUT:**

Adjacency Matrix:

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

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

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

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

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

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

=== Code Execution Successful ===

**BFS:**

**ADJ LIST:**

1. class GraphList:
2. def \_\_init\_\_(self):
3. self.graph={}
5. def add\_edge(self,u,v):
6. if u not in self.graph:
7. self.graph[u]=[]
8. if v not in self.graph:
9. self.graph[v]=[]
11. self.graph[u].append(v)
12. self.graph[v].append(u)
13. def display(self):
14. for i in self.graph:
15. print(f"{i} --> {self.graph[i]}")
16. def bfs(self,start):
17. visited=set()
18. queue=[start]
19. while queue:
20. vertex=queue.pop(0)
21. if vertex not in visited:
22. visited.add(vertex)
23. print(vertex,end=' ')
24. for i in self.graph[vertex]:
25. if i not in visited:
26. queue.append(i)
28. g=GraphList()
29. g.add\_edge(1,2)
30. g.add\_edge(1,3)
31. g.add\_edge(2,3)
32. g.add\_edge(2,4)
33. g.add\_edge(3,5)
34. g.add\_edge(5,7)
35. g.display()
36. g.bfs(1)

**OUTPUT:**

1 --> [2, 3]

2 --> [1, 3, 4]

3 --> [1, 2, 5]

4 --> [2]

5 --> [3, 7]

7 --> [5]

1 2 3 4 5 7

**BFS ADJ MATRIX**

1. class GraphAdjMatrix:
2. def \_\_init\_\_(self, size=None, adj\_matrix=None):
3. if adj\_matrix is not None:
4. self.adj\_matrix = adj\_matrix
5. self.size = len(adj\_matrix)
6. else:
7. self.size = size
8. self.adj\_matrix = [[0 for \_ in range(size)] for \_ in range(size)]
10. def add\_edge(self, u, v):
11. if u < self.size and v < self.size:
12. self.adj\_matrix[u][v] = 1
13. self.adj\_matrix[v][u] = 1
14. def display(self):
15. print("Adjacency Matrix:")
16. for row in self.adj\_matrix:
17. print(row)
18. def bfs(self, start):
19. visited = set()
20. queue = [start]
22. while queue:
23. vertex = queue.pop(0)
24. if vertex not in visited:
25. visited.add(vertex)
26. print(vertex, end=' ')
27. for i in range(self.size):
28. if self.adj\_matrix[vertex][i] == 1 and i not in visited and i not in queue:
29. queue.append(i)
30. adj\_matrix = [
31. [0, 1, 1, 0, 0, 0, 0],
32. [1, 0, 0, 1, 0, 0, 0],
33. [1, 0, 0, 1, 0, 0, 0],
34. [0, 1, 1, 0, 1, 0, 0],
35. [0, 0, 0, 1, 0, 1, 0],
36. [0, 0, 0, 0, 1, 0, 1],
37. [0, 0, 0, 0, 0, 1, 0]
38. ]
39. g = GraphAdjMatrix(adj\_matrix=adj\_matrix)
40. g.display()
41. print("BFS traversal starting from vertex 1:")
42. g.bfs(1)

**OUTPUT:**

Adjacency Matrix:

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

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

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

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

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

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

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

BFS traversal starting from vertex 1:

1 0 3 2 4 5 6

=== Code Execution Successful ===

**DFS ADJ LIST:**

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 i in self.graph:

print(f"{i} --> {self.graph[i]}")

def bfs(self, start):

visited = set()

queue = [start]

print("BFS Traversal:", end=' ')

while queue:

vertex = queue.pop(0)

if vertex not in visited:

visited.add(vertex)

print(vertex, end=' ')

for i in self.graph[vertex]:

if i not in visited and i not in queue:

queue.append(i)

print()

def dfs(self, start, visited=None):

if visited is None:

visited = set()

visited.add(start)

print(start, end=' ')

for i in self.graph[start]:

if i not in visited:

self.dfs(i, visited)

g = GraphList()

g.add\_edge(1, 2)

g.add\_edge(1, 3)

g.add\_edge(2, 3)

g.add\_edge(2, 4)

g.add\_edge(3, 5)

g.add\_edge(5, 7)

g.display()

g.bfs(1)

print("DFS Traversal:", end=' ')

g.dfs(5)

**OUTPUT:**

Adjacency List:

1 --> [2, 3]

2 --> [1, 3, 4]

3 --> [1, 2, 5]

4 --> [2]

5 --> [3, 7]

7 --> [5]

BFS Traversal: 1 2 3 4 5 7

DFS Traversal: 5 3 1 2 4 7

=== Code Execution Successful ===