**DFS using Graph**

from collections import deque

# BFS from given source s

def bfs(adj, s, visited):

q = deque() # Create a queue for BFS

# Mark the source node as visited and enqueue it

visited[s] = True

q.append(s)

# Iterate over the queue

while q:

curr = q.popleft() # Dequeue a vertex

print(curr, end=" ")

# Get all adjacent vertices of curr

for x in adj[curr]:

if not visited[x]:

visited[x] = True # Mark as visited

q.append(x) # Enqueue it

# Function to add an edge to the graph

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

adj[u].append(v)

adj[v].append(u) # Undirected graph

# Perform BFS for the entire graph

def bfs\_disconnected(adj):

visited = [False] \* len(adj) # Not visited

for i in range(len(adj)):

if not visited[i]:

bfs(adj, i, visited)

# Example usage

V = 6 # Number of vertices

adj = [[] for \_ in range(V)] # Adjacency list

# Add edges to the graph

add\_edge(adj, 0, 1)

add\_edge(adj, 0, 2)

add\_edge(adj, 3, 4)

add\_edge(adj, 4, 5)

# Perform BFS traversal for the entire graph

bfs\_disconnected(adj)

**DFS in Graph**

class Graph:

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

# Adjacency list

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

def add\_edge(self, s, t):

self.adj[s].append(t)

self.adj[t].append(s)

def dfs\_rec(self, visited, s):

visited[s] = True

print(s, end=" ")

# Recursively visit all adjacent vertices

# that are not visited yet

for i in self.adj[s]:

if not visited[i]:

self.dfs\_rec(visited, i)

def dfs(self):

visited = [False] \* len(self.adj)

# Loop through all vertices to handle disconnected

# graph

for i in range(len(self.adj)):

if not visited[i]:

# Perform DFS from unvisited vertex

self.dfs\_rec(visited, i)

if \_\_name\_\_ == "\_\_main\_\_":

V = 6 # Number of vertices

graph = Graph(V)

# Define the edges of the graph

edges = [(1, 2), (2, 0), (0, 3), (4, 5)]

# Populate the adjacency list with edges

for edge in edges:

graph.add\_edge(edge[0], edge[1])

print("Complete DFS of the graph:")

graph.dfs() # Perform DFS