class Graph:

def \_init\_(self, vertices):

self.vertices = vertices

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

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

self.adj\_list[u].append(v)

self.adj\_list[v].append(u)

def dfs(self, start):

#visited = [False] \* self.vertices

visited = []

for i in range(self.vertices):

visited.append(False)

self.\_dfs(start, visited)

def \_dfs(self, vertex, visited):

visited[vertex] = True

print(vertex, end=' ')

for neighbor in self.adj\_list[vertex]:

if not visited[neighbor]:

self.\_dfs(neighbor, visited)

def bfs(self, start):

visited = []

for i in range(self.vertices):

visited.append(False)

queue = []

queue.append(start)

visited[start] = True

while queue:

vertex = queue.pop(0)

print(vertex, end=' ')

for neighbor in self.adj\_list[vertex]:

if not visited[neighbor]:

visited[neighbor] = True

queue.append(neighbor)

# Example usage

vertices = int(input("Enter the number of vertices: "))

g = Graph(vertices)

edges = int(input("Enter the number of edges: "))

for i in range(edges):

u, v = map(int, input(f"Enter edge {i+1} (u, v): ").split())

g.add\_edge(u, v)

start = int(input("Enter the starting vertex for the search: "))

print("DFS Traversal:")

g.dfs(start)

print("\nBFS Traversal:")

g.bfs(start)