Exercise:11.1

class Graph:

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

self.V = vertices

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

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

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

self.matrix[u][v] = 1

self.matrix[v][u] = 1

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

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

def print\_matrix(self):

print("Adjacency Matrix:")

print(" ", end="")

for i in range(self.V):

print(f"{i} ", end="")

print()

for i in range(self.V):

print(f"{i}: ", end="")

for j in range(self.V):

print(f"{self.matrix[i][j]} ", end="")

print()

def print\_list(self):

print("Adjacency List:")

for i, neighbors in enumerate(self.adj\_list):

print(f"{i}: {' '.join(map(str, neighbors))}")

g = Graph(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.print\_matrix()

print()

g.print\_list()

output:

Adjacency Matrix:

0 1 2 3 4

0: 0 1 0 0 1

1: 1 0 1 1 1

2: 0 1 0 1 0

3: 0 1 1 0 1

4: 1 1 0 1 0

Adjacency List:

0: 1 4

1: 0 2 3 4

2: 1 3

3: 1 2 4

4: 0 1 3

Exercise:11.2

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

if visited is None:

visited = set()

visited.add(start)

print(start, end=' ')

for neighbor in graph[start]:

if neighbor not in visited:

dfs(graph, neighbor, visited)

def bfs(graph, start):

visited = set([start])

queue = [start] # Use list as queue

while queue:

vertex = queue.pop(0) # pop from front (inefficient but works)

print(vertex, end=' ')

for neighbor in graph[vertex]:

if neighbor not in visited:

visited.add(neighbor)

queue.append(neighbor)

# Example graph as adjacency list

graph = {

0: [1, 4],

1: [0, 2, 3, 4],

2: [1, 3],

3: [1, 2, 4],

4: [0, 1, 3]

}

print("DFS traversal starting at node 0:")

dfs(graph, 0)

print("\nBFS traversal starting at node 0:")

bfs(graph, 0)

output:

DFS traversal starting at node 0:

0 1 2 3 4

BFS traversal starting at node 0:

0 1 4 2 3