Bfs iterative

from collections import defaultdict

# BFS implementation

def bfs(graph, start):

visited = set()

queue = [start]

while queue:

vertex = queue.pop(0)

if vertex not in visited:

print(vertex) # or do whatever you want with the vertex

visited.add(vertex)

queue.extend(graph[vertex] - visited)

# Function to find path using BFS

def bfs\_path(graph, start, goal):

visited = set()

queue = [(start, [start])] # Queue holds tuples of (vertex, path)

while queue:

vertex, path = queue.pop(0)

if vertex not in visited:

if vertex == goal:

print("Path to goal:", path)

return path # Return the path to the goal

visited.add(vertex)

for neighbor in graph[vertex] - visited:

queue.append((neighbor, path + [neighbor]))

print("Goal not found")

return None # Return None if goal is not found

# Taking dynamic input for the graph

graph = defaultdict(set)

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

for \_ in range(num\_vertices):

vertex = input("Enter a vertex: ")

neighbors = input("Enter its neighbors (separated by spaces): ").split()

graph[vertex].update(neighbors)

start\_vertex = input("Enter the starting vertex: ")

goal\_vertex = input("Enter the goal vertex: ")

print("BFS traversal:")

bfs(graph, start\_vertex)

# Find and display path to the goal using DFS and BFS

print("BFS path to goal:")

bfs\_path(graph, start\_vertex, goal\_vertex)

# Enter the number of vertices: 5

# Enter a vertex: a

# Enter its neighbors (separated by spaces): b c e

# Enter a vertex: b

# Enter its neighbors (separated by spaces): a d

# Enter a vertex: c

# Enter its neighbors (separated by spaces): a d

# Enter a vertex: d

# Enter its neighbors (separated by spaces): a b c

# Enter a vertex: e

# Enter its neighbors (separated by spaces): a d

# Enter the starting vertex: a

# Enter the goal vertex: c

# BFS traversal:

# a

# e

# c

# b

# d

# BFS path to goal:

# Path to goal: ['a', 'c']