Dfs iterative

from collections import defaultdict

# DFS implementation

def dfs(graph, start):

visited = set()

stack = [start]

while stack:

vertex = stack.pop()

if vertex not in visited:

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

visited.add(vertex)

stack.extend(graph[vertex] - visited)

# Function to find path using DFS

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

visited = set()

stack = [(start, [start])] # Stack holds tuples of (vertex, path)

while stack:

vertex, path = stack.pop()

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:

stack.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: ")

# Perform DFS and BFS

print("DFS traversal:")

dfs(graph, start\_vertex)

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

print("DFS path to goal:")

dfs\_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 c d

# Enter a vertex: e

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

# Enter the starting vertex: a

# Enter the goal vertex: c

# DFS traversal:

# a

# c

# d

# e

# b

# DFS path to goal:

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