Dfs recursive  
from collections import defaultdict

def dfs\_recursive(graph, vertex, visited=None):

if visited is None:

visited = set()

visited.add(vertex)

print(vertex) # Do something with the vertex

for neighbor in graph[vertex]:

if neighbor not in visited:

dfs\_recursive(graph, neighbor, visited)

def dfs\_path\_recursive(graph, current, goal, path=None, visited=None):

if visited is None:

visited = set()

if path is None:

path = [current]

visited.add(current)

if current == goal:

return path

for neighbor in graph[current]:

if neighbor not in visited:

result\_path = dfs\_path\_recursive(graph, neighbor, goal, path + [neighbor], visited)

if result\_path: # If path found in recursion

return result\_path

return None # No path found

# Undirected graph input

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)

for neighbor in neighbors:

graph[neighbor].add(vertex) # Ensure undirected connection

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

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

print("\nRecursive DFS traversal:")

dfs\_recursive(graph, start\_vertex)

print("\nRecursive DFS path to goal:")

path = dfs\_path\_recursive(graph, start\_vertex, goal\_vertex)

if path:

print("Path to goal:", path)

else:

print("Goal not found")

# 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

# Recursive DFS traversal:

# a

# b

# d

# c

# e

# Recursive DFS path to goal:

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

* dfs\_recursive → Traverses and prints all reachable vertices from a start.
* dfs\_path\_recursive → Recursively finds and returns one path from start to goal.
* Graph is undirected (each connection goes both ways).
* Uses recursion and visited set to avoid cycles and repeated visits.