**ADS Assignment 4**

**Fast learner-Implement detection of connected component using DFS**

class Graph:

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

self.vertices = vertices

self.adjacency\_list = {}

for vertex in range(vertices):

self.adjacency\_list[vertex] = []

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

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

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

def dfs(self, vertex, visited, component):

visited[vertex] = True

component.append(vertex)

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

if not visited[neighbor]:

self.dfs(neighbor, visited, component)

def connected\_components(self):

visited = [False] \* self.vertices

components = []

for vertex in range(self.vertices):

if not visited[vertex]:

component = []

self.dfs(vertex, visited, component)

components.append(component)

return components

# Example Usage:

g = Graph(8)

g.add\_edge(0, 1)

g.add\_edge(1, 2)

g.add\_edge(3, 4)

g.add\_edge(5, 6)

g.add\_edge(6, 7)

connected\_components = g.connected\_components()

print("Connected Components:", connected\_components)

In this example, the Graph class represents an undirected graph, and the connected\_components method uses DFS to find the connected components.

Here's how the example graph looks like:

0 -- 1 3 -- 4

| |

2 5 -- 6 – 7

Output

Connected Components: [[0, 1, 2], [3, 4], [5, 6, 7]]

This output indicates that there are three connected components in the given graph: [0, 1, 2], [3, 4], and [5, 6, 7]. Each component is represented as a list of vertices.