|  |  |
| --- | --- |
|  | **Namal University Mianwali**  **Department of Computer Science** |

## **Lab 4**

# **Graphs**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Course** | **Name** | **Roll No** | **Submitted to** | **Department** |
| Artificial Intelligence | Usaira Shahbaz | NUM-BSCS-2022-27 | Mr. Ammar Ahmad | Computer Science, Namal University, Mianwali |

## **Task 1**

**CODE:**

*class Graph:*

*def \_\_init\_\_(self):*

*self.graph = {}*

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

*self.graph.setdefault(u, []).append(v)*

*self.graph.setdefault(v, []).append(u)*

*def display(self):*

*print("Graph Connectivity:")*

*for node, neighbors in self.graph.items():*

*print(f"{node} → {neighbors}")*

*def degrees(self):*

*print("\nDegree of Each Node:")*

*for node, neighbors in self.graph.items():*

*print(f"{node}: {len(neighbors)}")*

*graph = Graph()*

*edges = [(6, 4), (4, 5), (4, 3), (5, 1), (5, 2), (3, 2), (1, 2)]*

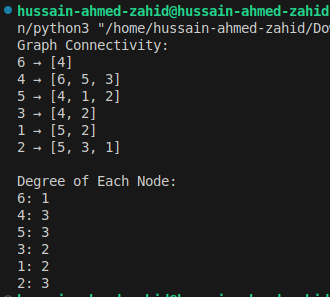
*for u, v in edges:*

*graph.add\_edge(u, v)*

*graph.display()*

*graph.degrees()*

**SCREENSHOT:**

**

## **Task 2**

**CODE:**

*class DirectedGraph:*

*def \_\_init\_\_(self):*

*self.graph = {}*

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

*self.graph.setdefault(u, []).append(v)*

*self.graph.setdefault(v, [])*

*def display(self):*

*print("Graph Connectivity:")*

*for node, neighbors in self.graph.items():*

*print(f"{node} → {neighbors}")*

*def in\_out\_degree(self):*

*in\_degree = {node: 0 for node in self.graph}*

*out\_degree = {node: len(neighbors) for node, neighbors in self.graph.items()}*

*for neighbors in self.graph.values():*

*for node in neighbors:*

*in\_degree[node] += 1*

*print("\nIn-Degree and Out-Degree of Each Node:")*

*for node in self.graph:*

*print(f"{node}: In-Degree = {in\_degree[node]}, Out-Degree = {out\_degree[node]}")*

*graph = DirectedGraph()*

*edges = [("A", "B"), ("B", "C"), ("B", "E"), ("B", "D"), ("D", "E"), ("C", "E"), ("E", "F"), ("G", "D")]*

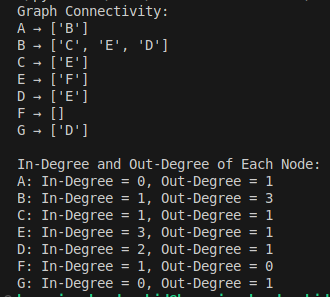
*for u, v in edges:*

*graph.add\_edge(u, v)*

*graph.display()*

*graph.in\_out\_degree()*

**SCREENSHOT:**



## **Task 3**

**CODE:**

*class Graph:*

*def \_\_init\_\_(self):*

*self.graph = {}*

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

*self.graph.setdefault(u, []).append(v)*

*self.graph.setdefault(v, []).append(u)*

*def find\_path(self, start, end, path=None):*

*if path is None:*

*path = []*

*path.append(start)*

*if start == end:*

*return path*

*for neighbor in self.graph.get(start, []):*

*if neighbor not in path:*

*new\_path = self.find\_path(neighbor, end, path.copy())*

*if new\_path:*

*return new\_path*

*return None*

*graph = Graph()*

*edges = [(6, 4), (4, 5), (4, 3), (5, 1), (5, 2), (3, 2), (1, 2)]*

*for u, v in edges:*

*graph.add\_edge(u, v)*

*path = graph.find\_path(6, 1)*

*print("Path from 6 to 1:", path if path else "No path found")*

**SCREENSHOT:**



## **Task 4**

**CODE:**

*class DirectedGraph:*

*def \_\_init\_\_(self):*

*self.graph = {}*

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

*self.graph.setdefault(u, []).append(v)*

*self.graph.setdefault(v, [])*

*def find\_path(self, start, end, path=None):*

*if path is None:*

*path = []*

*path.append(start)*

*if start == end:*

*return path*

*for neighbor in self.graph.get(start, []):*

*if neighbor not in path:*

*new\_path = self.find\_path(neighbor, end, path.copy())*

*if new\_path:*

*return new\_path*

*return None*

*graph = DirectedGraph()*

*edges = [("A", "B"), ("B", "C"), ("B", "E"), ("B", "D"), ("D", "E"), ("C", "E"), ("E", "F"), ("G", "D")]*

*for u, v in edges:*

*graph.add\_edge(u, v)*

*path = graph.find\_path("A", "F")*

*print("Path from A to F:", path if path else "No path found")*

**SCREENSHOT:**



## **Task 5**

**CODE:**

*class Graph:*

*def \_\_init\_\_(self):*

*self.graph = {}*

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

*self.graph.setdefault(u, []).append(v)*

*self.graph.setdefault(v, []).append(u)*

*def find\_all\_paths(self, start, end, path=None, all\_paths=None):*

*if path is None:*

*path = []*

*if all\_paths is None:*

*all\_paths = []*

*path.append(start)*

*if start == end:*

*all\_paths.append(path.copy())*

*else:*

*for neighbor in self.graph.get(start, []):*

*if neighbor not in path:*

*self.find\_all\_paths(neighbor, end, path.copy(), all\_paths)*

*return all\_paths*

*graph = Graph()*

*edges = [(6, 4), (4, 5), (4, 3), (5, 1), (5, 2), (3, 2), (1, 2)]*

*for u, v in edges:*

*graph.add\_edge(u, v)*

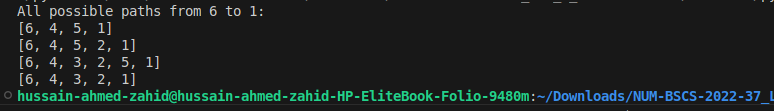
*all\_paths = graph.find\_all\_paths(6, 1)*

*print("All possible paths from 6 to 1:")*

*for path in all\_paths:*

*print(path)*

**SCREENSHOT:**



## **Task 6**

**CODE:**

*class DirectedGraph:*

*def \_\_init\_\_(self):*

*self.graph = {}*

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

*self.graph.setdefault(u, []).append(v)*

*self.graph.setdefault(v, [])*

*def find\_all\_paths(self, start, end, path=None, all\_paths=None):*

*if path is None:*

*path = []*

*if all\_paths is None:*

*all\_paths = []*

*path.append(start)*

*if start == end:*

*all\_paths.append(path.copy())*

*else:*

*for neighbor in self.graph.get(start, []):*

*if neighbor not in path:*

*self.find\_all\_paths(neighbor, end, path.copy(), all\_paths)*

*return all\_paths*

*graph = DirectedGraph()*

*edges = [("A", "B"), ("B", "C"), ("B", "E"), ("B", "D"), ("D", "E"), ("C", "E"), ("E", "F"), ("G", "D")]*

*for u, v in edges:*

*graph.add\_edge(u, v)*

*all\_paths = graph.find\_all\_paths("A", "F")*

*print("All possible paths from A to F:")*

*for path in all\_paths:*

*print(path)*

**SCREENSHOT:**

