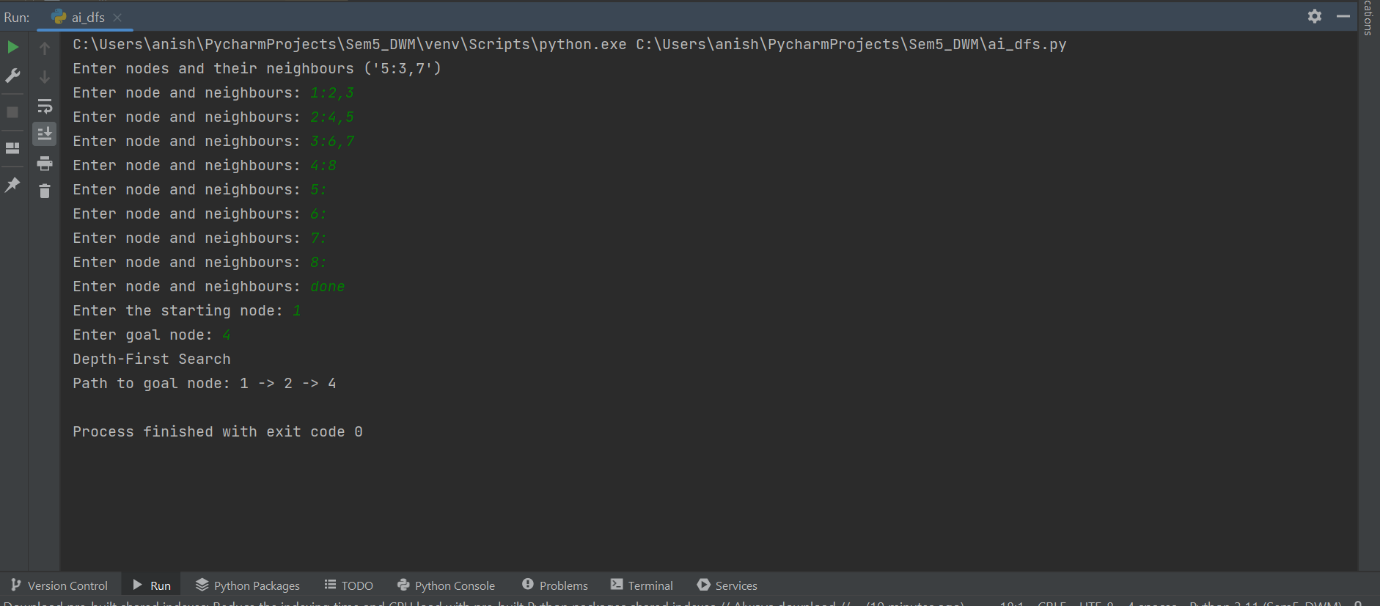
**Anisha Jain**

**C14 – 65**

**EXPERIMENT 2**

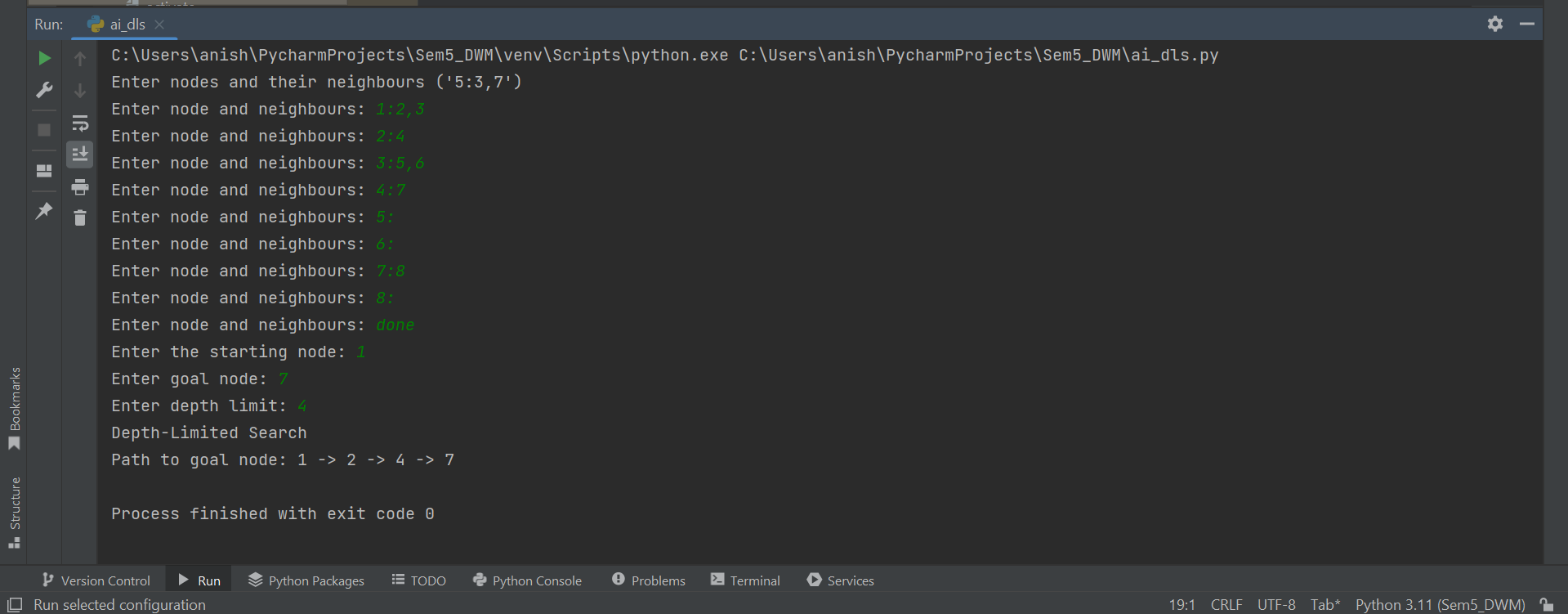
**DFS :**

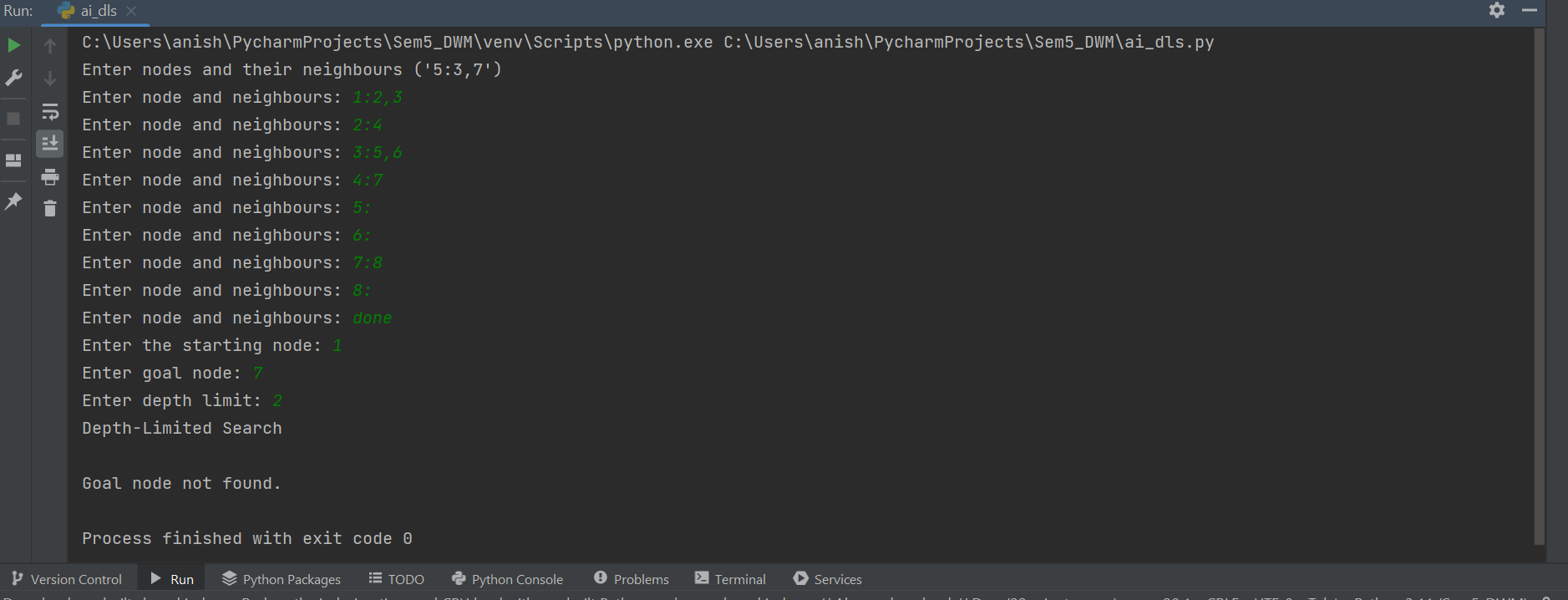
def dfs(visited, graph, node, goal, path): if node not in visited: path.append(node) visited.add(node) if node == goal: return True for neighbour in graph[node]: if dfs(visited, graph, neighbour, goal, path): return True path.pop() return Falsegraph = {}print("Enter nodes and their neighbours ('5:3,7')")while True: node\_input = input("Enter node and neighbours: ").strip() if node\_input.lower() == 'done': break node, neighbours = node\_input.split(':') if neighbours: graph[node] = neighbours.split(',') else: graph[node] = []visited = set()path = []start\_node = input("Enter the starting node: ")goal\_node = input("Enter goal node: ")print("Depth-First Search")if start\_node == goal\_node: print("Start and goal nodes are the same.")elif dfs(visited, graph, start\_node, goal\_node, path): print("Path to goal node:", ' -> '.join(path))else: print("Goal node not found.")



**DLS :**

def dls(visited, graph, node, goal, path, depth, depth\_limit): if depth > depth\_limit: return False if node not in visited: path.append(node) visited.add(node) if node == goal: return True for neighbour in graph[node]: if dls(visited, graph, neighbour, goal, path, depth + 1, depth\_limit): return True path.pop() return Falsegraph = {}print("Enter nodes and their neighbours ('5:3,7')")while True: node\_input = input("Enter node and neighbours: ").strip() if node\_input.lower() == 'done': break node, neighbours = node\_input.split(':') if neighbours: graph[node] = neighbours.split(',') else: graph[node] = []visited = set()path = []start\_node = input("Enter the starting node: ")goal\_node = input("Enter goal node: ")depth\_limit = int(input("Enter depth limit: "))print("Depth-Limited Search")if dls(visited, graph, start\_node, goal\_node, path, 0, depth\_limit): print("Path to goal node:", ' -> '.join(path))else: print("\nGoal node not found.")





**DFID :**

def iddfs(graph, node, goal, depth, visited=None, path=None): if visited is None: visited = set() if path is None: path = [] visited.add(node) path.append(node) print("Visiting node:", node) # Print visiting node if depth == 0 and node == goal: return True, path if depth > 0: for neighbor in graph.get(node, []): if neighbor not in visited: found, current\_path = iddfs(graph, neighbor, goal, depth - 1, visited, path) if found: return True, current\_path path.pop() return False, pathdef main(): graph = { 'S': ['A', 'B'], 'A': ['S', 'C', 'D'], 'B': ['S', 'I', 'J'], 'C': ['A', 'E', 'F'], 'D': ['A', 'G'], 'E': ['C'], 'F': ['C'], 'G': ['D'], 'I': ['B', 'H'], 'J': ['B'], 'H': ['I'] } start\_node = 'S' goal\_node = 'J' max\_depth = 2 for depth in range(max\_depth + 1): print(f"Iteration: {depth+1}") print(f"Depth Level: {depth}") found, path = iddfs(graph, start\_node, goal\_node, depth) if found: print(f"Goal node '{goal\_node}' found starting from node '{start\_node}' using IDDFS at depth {depth}.") print("Path traversed:", ' -> '.join(path)) break else: print(f"Goal node '{goal\_node}' not found at depth {depth}.") print()if \_\_name\_\_ == "\_\_main\_\_": main()

