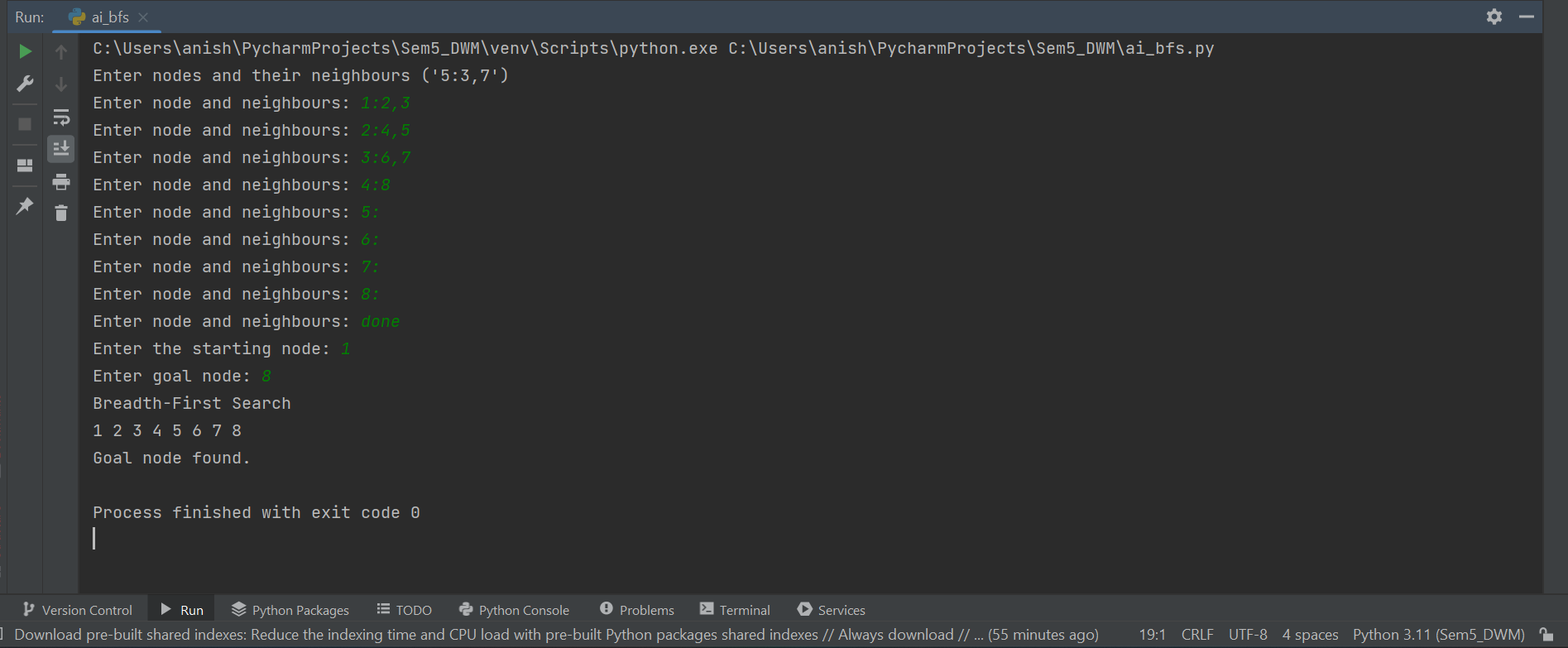
**Anisha Jain**

**C14 – 65**

**EXPERIMENT 3**

**BFS :**

def bfs(visited, graph, node, goal): visited.append(node) queue.append(node) while queue: m = queue.pop(0) print(m, end=" ") if m == goal: return True for neighbour in graph[m]: if neighbour not in visited: visited.append(neighbour) queue.append(neighbour) 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 = []queue = []start\_node = input("Enter the starting node: ")goal\_node = input("Enter goal node: ")print("Breadth-First Search")if bfs(visited, graph, start\_node, goal\_node): print("\nGoal node found.")else: print("\nGoal node not found.")



**UCS :**

import heapqdef ucs(graph, start\_node, goal\_node): visited = set() priority\_queue = [(0, start\_node, [])] while priority\_queue: cost, node, path = heapq.heappop(priority\_queue) if node not in visited: visited.add(node) path.append(node) if node == goal\_node: print("Path to goal node:", ' -> '.join(path)) print("Total Cost:", cost) return for neighbour, edge\_cost in graph[node]: if neighbour not in visited: heapq.heappush(priority\_queue, (cost + edge\_cost, neighbour, path[:])) print("Goal node not found.") return None# Define the graphgraph = { 'A': [('B', 4), ('C', 2)], 'B': [('D', 5), ('E', 10)], 'C': [('F', 3)], 'D': [('G', 5)], 'E': [], 'F': [], 'G': []

}start\_node = input("Enter the starting node: ")goal\_node = input("Enter the goal node: ")ucs(graph, start\_node, goal\_node)

