**Anisha Jain**

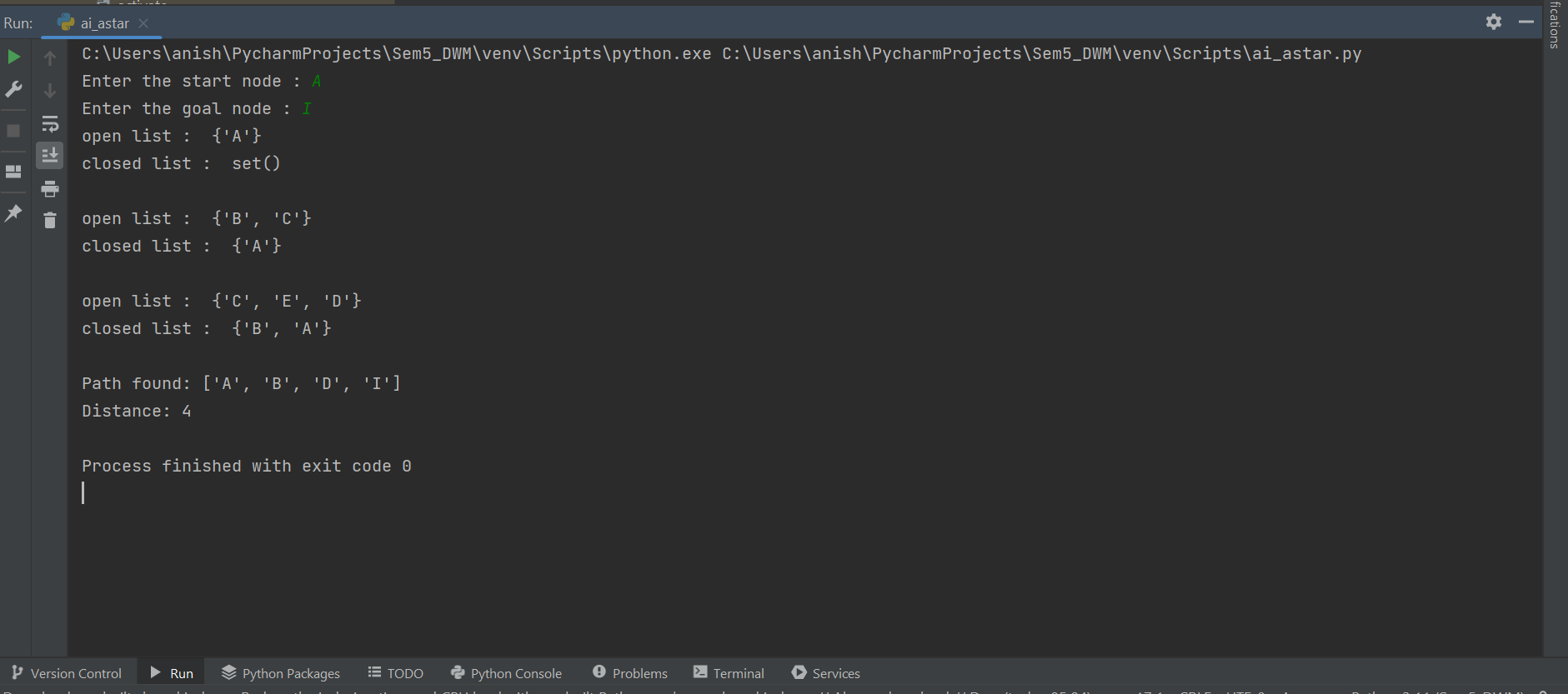
**C14 – 65**

**EXPERIMENT 4**

**A\* Algorithm :**

def aStarAlgo(start\_node, stop\_node): open\_set = set([start\_node]) closed\_set = set() g = {start\_node: 0} parents = {start\_node: start\_node} def get\_neighbors(v): if v in Graph\_nodes: return Graph\_nodes[v] else: return None def heuristic(n): H\_dist = { 'A': 11, 'B': 6, 'C': 99, 'D': 1, 'E': 7, 'F': 2, 'G': 0, 'H': 2, 'I': 0

} return H\_dist[n] while len(open\_set) > 0: n = None for v in open\_set: if n is None or heuristic(v) + g[v] < heuristic(n) + g[n]: n = v if n == stop\_node or Graph\_nodes[n] is None: pass else: print("open list : ", open\_set) print("closed list : ", closed\_set) print() for (m, weight) in get\_neighbors(n): if m not in open\_set and m not in closed\_set: open\_set.add(m) parents[m] = n g[m] = g[n] + weight else: if g[m] > g[n] + weight: g[m] = g[n] + weight parents[m] = n if m in closed\_set: closed\_set.remove(m) open\_set.add(m) if n is None: print('Path does not exist!') return None if n == stop\_node: path = [] while parents[n] != n: path.append(n) n = parents[n] path.append(start\_node) path.reverse() print('Path found:', path) print('Distance:', g[stop\_node]) return path open\_set.remove(n) closed\_set.add(n) print('Path does not exist!') return NoneGraph\_nodes = { 'A': [('B', 2), ('C', 3)], 'B': [('D', 1), ('E', 9)], 'C': [('F', 3)], 'E': [('H', 6)], 'D': [('I', 1)], 'F': None, 'G': None, 'H': None, 'I': None}start = str(input("Enter the start node : "))goal = str(input("Enter the start node : "))aStarAlgo(start, goal)



**Greedy Best First Search :**

def greedyBestFirstSearch(start\_node, stop\_node): open\_set = set([start\_node]) closed\_set = set() parents = {start\_node: start\_node} total\_heuristic = {start\_node: 0} # Keep track of total heuristic path used def get\_neighbors(v): if v in Graph\_nodes: return Graph\_nodes[v] else: return None def heuristic(n): H\_dist = { 'A': 11, 'B': 6, 'C': 99, 'D': 1, 'E': 7, 'F': 2, 'G': 0, 'H': 2, 'I': 0 } return H\_dist[n] while len(open\_set) > 0: n = None print("Open List : ", open\_set) print("Closed List : ", closed\_set) print() for v in open\_set: if n is None or heuristic(v) < heuristic(n): n = v if n is None: print('Path does not exist!') return None if n == stop\_node: path = [] while parents[n] != n: path.append(n) n = parents[n] path.append(start\_node) path.reverse() print('Path found:', path) print('Total Heuristic Path:', total\_heuristic[stop\_node]) return path open\_set.remove(n) closed\_set.add(n) if Graph\_nodes[n] is not None: for (m, \_) in get\_neighbors(n): if m not in open\_set and m not in closed\_set: open\_set.add(m) parents[m] = n total\_heuristic[m] = total\_heuristic[n] + heuristic(m) # Update total heuristic path used print('Path does not exist!') return NoneGraph\_nodes = { 'A': [('B', 2), ('C', 3)], 'B': [('D', 1), ('E', 9)], 'C': [('F', 3)], 'E': [('G', 6)], 'D': [('I', 1)], 'F': None, 'G': None, 'H': None, 'I': None}start = str(input("Enter the start node : "))goal = str(input("Enter the goal node : "))greedyBestFirstSearch(start, goal)

****