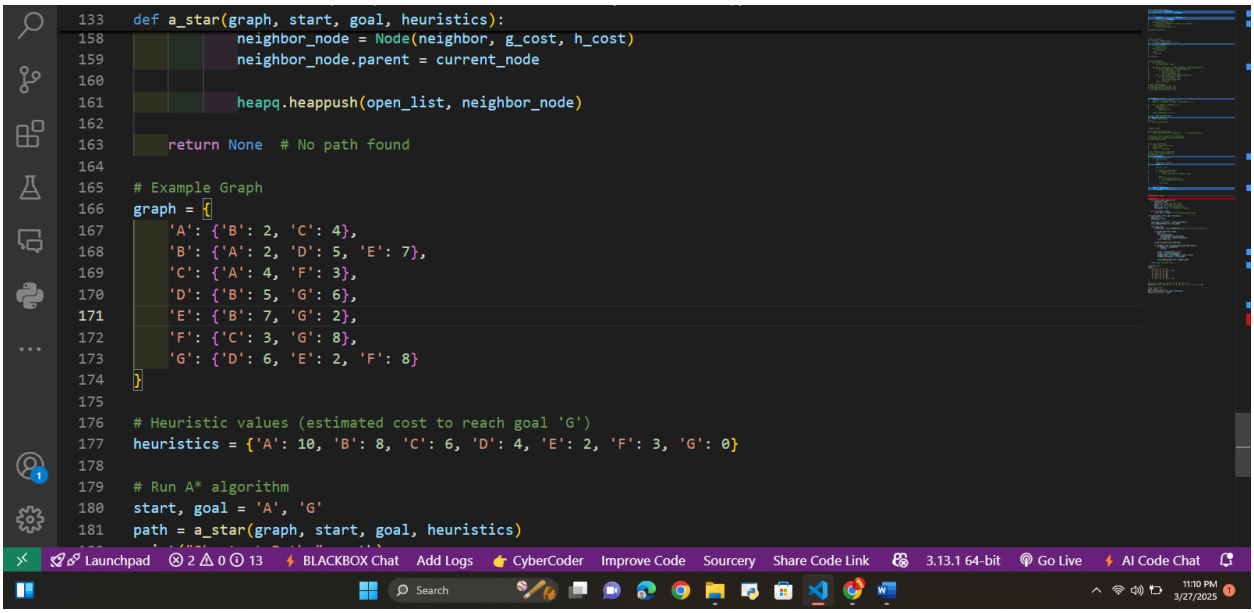
TASK 7

Name: Sami Imran

Roll No: 033



```
def a_star(graph, start, goal, heuristics):
            neighbor_node = Node(neighbor, g_cost, h_cost)
            neighbor_node.parent = current_node

            heapq.heappush(open_list, neighbor_node)

    return None  # No path found

# Example Graph
graph = {
    'A': {'B': 2, 'C': 4},
    'B': {'A': 2, 'D': 5, 'E': 7},
    'C': {'A': 4, 'F': 3},
    'D': {'B': 5, 'G': 6},
    'E': {'B': 7, 'G': 2},
    'F': {'C': 3, 'G': 8},
    'G': {'D': 6, 'E': 2, 'F': 8}
}

# Heuristic values (estimated cost to reach goal 'G')
heuristics = {'A': 10, 'B': 8, 'C': 6, 'D': 4, 'E': 2, 'F': 3, 'G': 0}

# Run A* algorithm
start, goal = 'A', 'G'
path = a_star(graph, start, goal, heuristics)
```

The *A algorithm** is an efficient and widely used pathfinding algorithm that helps find the shortest path between a start and a goal node in a graph. It works by evaluating nodes based on a cost function, which is the sum of two values: **g(n)**, the actual cost from the start node to the current node, and **h(n)**, the estimated heuristic cost from the current node to the goal. The algorithm uses a **priority queue** (min-heap) to always explore the node with the lowest total cost **f(n) = g(n) + h(n)**, ensuring an optimal path. It begins by adding the start node to an open list, processes the node with the lowest f(n), expands its neighbors, and continues until the goal is reached. If a shorter path to a node is found, the algorithm updates the path and recalculates costs. A* is widely used in **AI, robotics, navigation systems, and games** because of its ability to find the shortest path efficiently while balancing speed and accuracy. The heuristic function plays a crucial role, guiding the search towards the goal and preventing unnecessary computations. If the

heuristic is **admissible** (never overestimates the actual cost), A* guarantees finding the **shortest path**.