```
# A* Search: Consider the following graph where nodes represent cities, and edges represent the cost (distance) of traveling between them
# The heuristic values (straight-line distance to the goal) are given in the table.


# ----------------------------
# Q1: A* Search Implementation
# ----------------------------
import heapq

def a_star_search(graph, heuristics, start, goal):
    # Priority queue: (f(n), g(n), node, path)
    frontier = [(heuristics[start], 0, start, [start])]
    explored = {}

    print("A* Search Steps:")
    while frontier:
        f, g, node, path = heapq.heappop(frontier)
        print(f"Expanding {node}: g={g}, h={heuristics[node]}, f={f}, path={path}")

        if node == goal:
            print("\nOptimal Path Found!")
            return path, g

        if node in explored and explored[node] <= g:
            continue
        explored[node] = g

        for neighbor, cost in graph.get(node, []):
            g_new = g + cost
            f_new = g_new + heuristics[neighbor]
            heapq.heappush(frontier, (f_new, g_new, neighbor, path + [neighbor]))

    return None, float('inf')

# Graph for A* search
astar_graph = {
    'S': [('A', 1), ('B', 4)],
    'A': [('C', 2), ('D', 5)],
    'B': [('D', 1), ('E', 6)],
    'C': [('G', 7)],
    'D': [('G', 3)],
    'E': [('G', 2)]
}

astar_heuristics = {
    'S': 10, 'A': 8, 'B': 7, 'C': 4, 'D': 2, 'E': 3, 'G': 0
}

path, cost = a_star_search(astar_graph, astar_heuristics, 'S', 'G')
print("Final Path:", path, "with Cost:", cost)


# ----------------------------------------
# Q2: Greedy Best-First Search Implementation
# ----------------------------------------
def greedy_best_first(graph, heuristics, start, goal):
    frontier = [(heuristics[start], start, [start], 0)]
    visited = set()

    print("\nGreedy Best-First Search Steps:")
    while frontier:
        frontier.sort(key=lambda x: x[0])  # pick node with smallest h(n)
        h, node, path, cost = frontier.pop(0)
        print(f"Expanding {node}: h={h}, path={path}, cost={cost}")

        if node == goal:
            print("\nPath Found!")
            return path, cost

        visited.add(node)

        for neighbor, edge_cost in graph.get(node, []):
            if neighbor not in visited:
                frontier.append((heuristics[neighbor], neighbor, path + [neighbor], cost + edge_cost))

    return None, float('inf')

# Graph for GBFS
gbfs_graph = {
    'S': [('A', 2), ('B', 1)],
    'A': [('C', 2), ('D', 2)],
    'B': [('E', 2), ('F', 7)]
```

```
        'B': [('E', 2), ('F', 7)],
        'C': [('G', 6)],
        'D': [('G', 2)],
        'E': [('G', 5)],
        'F': [('G', 1)]
    }

    gbfs_heuristics = {
        'S': 6, 'A': 4, 'B': 2, 'C': 6, 'D': 1, 'E': 3, 'F': 2, 'G': 0
    }

    path, cost = greedy_best_first(gbfs_graph, gbfs_heuristics, 'S', 'G')
    print("Final Path:", path, "with Cost:", cost)
    print("\nNote: GBFS is not guaranteed to find the optimal path because it ignores g(n) (actual cost so far). It only uses h(n).\n")
```

```
A* Search Steps:
Expanding S: g=0, h=10, f=10, path=['S']
Expanding A: g=1, h=8, f=9, path=['S', 'A']
Expanding C: g=3, h=4, f=7, path=['S', 'A', 'C']
Expanding D: g=6, h=2, f=8, path=['S', 'A', 'D']
Expanding G: g=9, h=0, f=9, path=['S', 'A', 'D', 'G']

Optimal Path Found!
Final Path: ['S', 'A', 'D', 'G'] with Cost: 9

Greedy Best-First Search Steps:
Expanding S: h=6, path=['S'], cost=0
Expanding B: h=2, path=['S', 'B'], cost=1
Expanding F: h=2, path=['S', 'B', 'F'], cost=8
Expanding G: h=0, path=['S', 'B', 'F', 'G'], cost=9

Path Found!
Final Path: ['S', 'B', 'F', 'G'] with Cost: 9

Note: GBFS is not guaranteed to find the optimal path because it ignores g(n) (actual cost so far). It only uses h(n).
```