In [2]:

```python
def ASTAR(graph, start, goal, heuristic):
    openlist = [(0 + heuristic[start], start, [start])]  # Initialize the open list with the start node
    gcost = {start: 0}  # Initialize g-cost for the start node as 0
    parent = {}  # Dictionary to store the parent of each node for path reconstruction

    while openlist:
        openlist.sort(key=lambda x: x[0])  # Sort open list by the f-cost (g + h)
        _, currentnode, path = openlist.pop(0)  # Pop the node with the lowest f-cost

        if currentnode == goal:  # If the goal node is reached
            totalcost = gcost[goal]  # Get the total g-cost to the goal
            print("Path found: ", path)
            print("Total cost: ", totalcost)
            return path, totalcost  # Return the path and its total cost

        # Explore all neighbors of the current node
        for neighbour, cost in graph.get(currentnode, []):
            tentativegcost = gcost[currentnode] + cost  # Calculate tentative g-cost

            # If the neighbor has not been visited or a shorter path is found
            if neighbour not in gcost or tentativegcost < gcost[neighbour]:
                gcost[neighbour] = tentativegcost  # Update g-cost for the neighbor
                fcost = tentativegcost + heuristic[neighbour]  # Calculate f-cost (g + h)
                parent[neighbour] = currentnode  # Set the current node as the parent of the neighbor

                # Add the neighbor to the open list with its f-cost, node, and path
                openlist.append((fcost, neighbour, path + [neighbour]))

    return None, None  # If the goal is not reachable, return None

# Graph and heuristic values
graph = {
    'A': [('B', 1), ('C', 3)],
    'B': [('A', 1), ('D', 2)],
    'C': [('A', 3), ('D', 1)],
    'D': [('B', 2), ('C', 1), ('E', 4)],
    'E': [('D', 4)]
}

heuristic = {
    'A': 7, 'B': 6, 'C': 2, 'D': 1, 'E': 0
}

path, totalcost = ASTAR(graph, 'A', 'E', heuristic)
```

```
Path found:  ['A', 'B', 'D', 'E']
Total cost:  7
```