In [3]:

```
# week -7
import heapq

class Graph:
    def __init__(self):
        self.graph = {}

    def add_edge(self, u, v, cost):
        if u not in self.graph:
            self.graph[u] = []
        self.graph[u].append((v, cost))

    def a_star(self, start, goal, heuristic):
        open_list = []
        heapq.heappush(open_list, (0 + heuristic[start], start))
        g_cost = {start: 0}
        f_cost = {start: heuristic[start]}
        parent = {start: None}

        while open_list:
            current_node = heapq.heappop(open_list)[1]

            if current_node == goal:
                path = []
                while current_node is not None:
                    path.append(current_node)
                    current_node = parent[current_node]
                path.reverse()
                return path

            for neighbor, cost in self.graph.get(current_node, []):
                tentative_g_cost = g_cost[current_node] + cost
                if neighbor not in g_cost or tentative_g_cost < g_cost[neighbor
                    g_cost[neighbor] = tentative_g_cost
                    f_cost[neighbor] = tentative_g_cost + heuristic.get(neighbo
                    parent[neighbor] = current_node
                    heapq.heappush(open_list, (f_cost[neighbor], neighbor))

        return None

if __name__ == "__main__":
    g = Graph()
    g.add_edge("A", "B", 1)
    g.add_edge("A", "C", 4)
    g.add_edge("B", "D", 2)
    g.add_edge("C", "D", 5)
    g.add_edge("D", "E", 3)

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

    start = "A"
    goal = "E"
    path = g.a_star(start, goal, heuristic)

    if path:
        print("Path found: ", path)
    else:
        print("No path found")
```

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

In [6]:

```
#week  -6
class Node:
    def __init__(self, state, parent=None, h=0):
        self.state = state
        self.parent = parent
        self.h = h

    def __lt__(self, other):
        return self.h < other.h

def greedy_best_first_search(start, goal, heuristic):
    open_list = [Node(start, None, heuristic[start])]
    closed_list = set()

    while open_list:
        open_list.sort()
        current_node = open_list.pop(0)

        if current_node.state == goal:
            path = []
            while current_node:
                path.append(current_node.state)
                current_node = current_node.parent
            return path[::-1]

        closed_list.add(current_node.state)
        neighbors = get_neighbors(current_node.state)

        for neighbor in neighbors:
            if neighbor not in closed_list:
                open_list.append(Node(neighbor, current_node, heuristic[neighbo

    return None

def get_neighbors(state):
    return ["B", "C", "D"]

heuristic = {"A": 6, "B": 5, "C": 4, "D": 3, "E": 2, "F": 0}
start = "A"
goal = "F"
path = greedy_best_first_search(start, goal, heuristic)
print("Path found:", path)
```

```
Path found: None
```

In [ ]: