Name of Student : AHMED ALI ANSARI **ID No : 1402-2020**

Task :

1. **Implementing A* for heuristic search (Informed Search) ?**

Answer :

In [1]:
```
from queue import PriorityQueue

def heuristic(node, goal):
    """
    Heuristic function to estimate the cost from a given node to the goal.
    You need to define your own heuristic based on the problem domain.
    """
    # Implement your heuristic function here
    return 0

def a_star_search(graph, start, goal):
    """
    A* search algorithm to find the optimal path from the start node to the goal node.
    `graph` is a dictionary representing the graph, where the keys are the nodes and the values are dictionaries
    containing the neighboring nodes and their corresponding edge costs.
    `start` and `goal` are the start and goal nodes, respectively.
    """
    # Priority queue to store the nodes to be explored, ordered by the cost
    # f(n) = g(n) + h(n), where g(n) is the cost to reach the node and h(n) is the estimated cost to the goal
    frontier = PriorityQueue()
    frontier.put((0, start))

    # Dictionary to keep track of the cost to reach each node
    cost_so_far = {start: 0}

    # Dictionary to keep track of the parent node for each node
    parent = {start: None}
```

Name of Student : AHMED ALI ANSARI **ID No : 1402-2020**

```
    while not frontier.empty():
        _, current = frontier.get()

        if current == goal:
            break

        for neighbor in graph[current]:
            # Calculate the new cost to reach the neighbor node
            new_cost = cost_so_far[current] + graph[current][neighbor]
            if neighbor not in cost_so_far or new_cost < cost_so_far[neighbor]:
                cost_so_far[neighbor] = new_cost
                priority = new_cost + heuristic(neighbor, goal)
                frontier.put((priority, neighbor))
                parent[neighbor] = current

    # Reconstruct the path from the goal to the start
    path = []
    while current:
        path.append(current)
        current = parent[current]
    path.reverse()

    return path

# Example usage
# Define the graph
```

```
graph = {
    'A': {'B': 5, 'C': 3},
    'B': {'D': 2},
    'C': {'D': 4, 'E': 6},
    'D': {'E': 1, 'F': 7},
    'E': {'F': 3},
    'F': {}
}

start_node = 'A'
goal_node = 'F'

# Find the optimal path using A* search
optimal_path = a_star_search(graph, start_node, goal_node)
print(optimal_path)
```

```
['A', 'C', 'D', 'E', 'F']
```

Name of Student : **AHMED ALI ANSARI** **ID No :** **1402-2020**

2. Implementing Breadth First search from heuristic search (Un-Informed Search).?

Answer :

```
In [2]: from collections import deque

def bfs(graph, start, goal):
    # Create a queue for BFS
    queue = deque()
    # Enqueue the start node and mark it as visited
    queue.append(start)
    visited = set()
    visited.add(start)

    # Keep track of the paths
    paths = {start: []}

    while queue:
        # Dequeue a node from the queue
        node = queue.popleft()

        # Check if the goal node is found
        if node == goal:
            return paths[node]

        # Explore the neighbors of the current node
        for neighbor in graph[node]:
            if neighbor not in visited:
                # Enqueue the neighbor if it's not visited
                queue.append(neighbor)
                visited.add(neighbor)
                # Update the path to the neighbor
                paths[neighbor] = paths[node] + [neighbor]

    # If goal node is not found
    return None

# Example usage
graph = {
    'A': ['B', 'C'],
    'B': ['D', 'E'],
    'C': ['F'],
    'D': [],
    'E': ['F'],
    'F': []
}
start_node = 'A'
goal_node = 'F'

path = bfs(graph, start_node, goal_node)

if path:
    print(f"Path from {start_node} to {goal_node}: {path}")
else:
    print(f"No path found from {start_node} to {goal_node}")
```

```
Path from A to F: ['C', 'F']
```