Representation of Search Problem

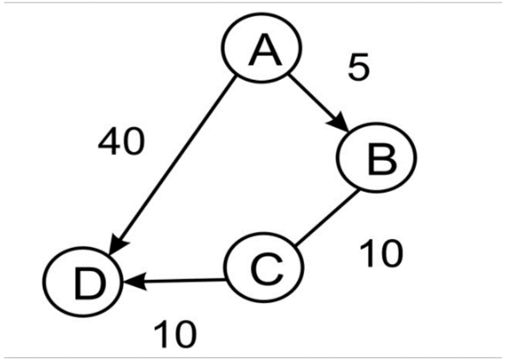
The problem-solving agent performs precisely by defining problems and its several solutions. The main components of problem formulation are as follows:

- **Goal Formulation:** It organizes the steps/sequence required to formulate one goal out of multiple goals as well as actions to achieve that goal. Goal formulation is based on the current situation and the agent's performance measure.
- **Initial State:** It is the starting state or initial step of the agent towards its goal.
- **Actions:** It is the description of the possible actions available to the agent.
- **Transition Model:** It describes what each action does.
- **Goal Test:** It determines if the given state is a goal state.
- **Path cost:** It assigns a numeric cost to each path that follows the goal. The problem-solving agent selects a cost function, which reflects its performance measure. Remember, an optimal solution has the lowest path cost among all the solutions.

Explicit Representation of the Search Graph:

The first representation of a search problem is from an explicit graph (as opposed to one that is generated as needed). An explicit graph consists of

- a list or set of nodes
- a list or set of arcs
- a start node
- list or set of goal nodes



```python
# Graph without weights
MyGraph = {
            'A':['B','D'],
            'B':['C'],
            'C':['B','D'],
          }
# Graph with weights
MyGraph = {
            'A':[('B',5),('D',40)],
            'B':[('C',10)],
            'C':[('B',10),('D',10)],
          }
print(MyGraph)
```

```
{'A': [('B', 5), ('D', 40)], 'B': [('C', 10)], 'C': [('B', 10), ('D', 10)]}
```

```python
# Add a vertex to the dictionary
def add_vertex(v):
    global graph
    global vertices_no
    if v in graph:
        print("Vertex ", v, " already exists.")
    else:
        vertices_no = vertices_no + 1
        graph[v] = []

# Add an edge between vertex v1 and v2 with edge weight e
def add_edge(v1, v2, e):
    global graph
    # Check if vertex v1 is a valid vertex
    if v1 not in graph:
        print("Vertex ", v1, " does not exist.")
        # Check if vertex v2 is a valid vertex
    elif v2 not in graph:
        print("Vertex ", v2, " does not exist.")
    else:
        # Since this code is not restricted to a directed or
        # an undirected graph, an edge between v1 v2 does not
        # imply that an edge exists between v2 and v1
        temp = [v2, e]
        graph[v1].append(temp)
```

```python
# Print the graph
def print_graph():
    global graph
    for vertex in graph:
        for edges in graph[vertex]:
            print(vertex, " -> ", edges[0], " edge weight: ", edges[1])

# driver code
graph = {}
# stores the number of vertices in the graph
vertices_no = 0
add_vertex('A')
add_vertex('B')
add_vertex('C')
add_vertex('D')
# Add the edges between the vertices by specifying
# the from and to vertex along with the edge weights.
add_edge('A', 'B', 5)
add_edge('A', 'D', 40)
add_edge('B', 'C', 10)
add_edge('C', 'B', 10)
add_edge('C', 'D', 10)
print_graph()
# Reminder: the second element of each list inside the dictionary
# denotes the edge weight.
print ("Internal representation: ", graph)
```

```
A  ->  B  edge weight:  5
A  ->  D  edge weight:  40
B  ->  C  edge weight:  10
C  ->  B  edge weight:  10
C  ->  D  edge weight:  10
Internal representation:  {'A': [['B', 5], ['D', 40]], 'B': [['C', 10]], 'C': [['B', 10], ['D', 10]], 'D': []}
```