Job No:05

Job Name: **Write a program for Traveling Salesman Problem**

Theory: The travelling salesman problem is a graph computational problem. The goal is to find the shortest possible route (Hamiltonian cycle) that visits all the cities exactly once and returns to the starting city, using the distances between cities as weights on the edges of the graph

Code:

```
def tsp(graph):
    num_nodes = len(graph)
    visited = [False] * num_nodes
    visited[0] = True  # Starting node
    min_cost = float("inf")
    optimal_path = []

    def dfs(current_node, path, cost, count):
        nonlocal min_cost, optimal_path
      if count == num_nodes and graph[current_node][0] != 0:
            cost += graph[current_node][0]
            if cost < min_cost:
                min_cost = cost
                optimal_path = path[:]  # Make a copy of the path
                optimal_path.append(1)
            return

        for next_node in range(num_nodes):
            if not visited[next_node] and graph[current_node][next_node] != 0:
                visited[next_node] = True
                path.append(next_node + 1)
                dfs(next_node,path,cost + graph[current_node][next_node],count+1)
                path.pop()  # Remove the node from the path
                visited[next_node] = False

    dfs(0, [1], 0, 1)  # Start with node 1, cost 0, and count 1
 return min_cost, optimal_path
graph = [
    [0, 10, 15, 20],
    [5, 0, 25, 10],
    [15, 30, 0, 5],
    [15, 10, 20, 0],
]
result = tsp(graph)
print("Minimum cost:", result[0])
print("Optimal path:", " -> ".join(map(str, result[1])))
```

Input/Output:

```
Minimum cost: 35
Optimal path: 1 -> 3 -> 4 -> 2 -> 1
```

Job No:06

Job Name: Write a program for Uniform Cost Search

Theory: Uniform Cost Search (UCS) is an uninformed search algorithm to find the lowest-cost path from a starting node to a goal node in a weighted graph. It explores the search space based solely on the information available in the graph, without using domain-specific knowledge. UCS guarantees the optimal solution in terms of the minimum cost but may be less efficient compared to informed search algorithms in certain scenarios.

Code:

```
def uniform_cost_search(tree, goal_node):
    priority_queue = []
    priority_queue.append({"node": tree, "path": [], "cost": 0})

    while priority_queue:
        min_cost_index = 0
        for i in range(1, len(priority_queue)):
            if priority_queue[i]["cost"] <
priority_queue[min_cost_index]["cost"]:
                min_cost_index = i
       current = priority_queue.pop(min_cost_index)
        node, path, cost = current["node"], current["path"], current["cost"]

        if node["value"] == goal_node:
            return {"cost": cost, "path": path + [node["value"]]}
      for child in node["children"]:
            cost_to_child = child["cost"]
            total_cost = cost + cost_to_child
            priority_queue.append(
                {
                    "node": child,
                    "path": path + [node["value"]],
                    "cost": total_cost,
                }
            )

    return None  # Goal not reachable

tree = {
    "value": "A",
    "cost": 0,
    "children": [
        {
```

```
            "value": "B",
            "cost": 2,
            "children": [
                {
                    "value": "D",
                    "cost": 6,
                    "children": [],
                },
                 {
                    "value": "E",
                    "cost": 4,
                    "children": [],
                },
            ],
        },
        {
            "value": "C",
            "cost": 3,
            "children": [
                {
                    "value": "F",
                    "cost": 4,
                    "children": [],
                },
            ],
        },
    ],
}

goal_node = "E"
result = uniform_cost_search(tree, goal_node)

if result is not None:
    print("Shortest Path:", " -> ".join(result["path"]))
    print("Total Cost:", result["cost"])
else:
    print("Goal not reachable.")
```

Input/Output:

```
Shortest Path: A -> B -> E
Total Cost: 6
```