Job No: 05

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

Theory:

The Traveling Salesman Problem (TSP) is the problem of finding the shortest possible route or path that visits a given set of cities exactly once and returns to the starting city.

It can be visualized as a salesman trying to find the shortest route to visit multiple cities and return to the starting city to minimize travel distance.

Code:

```
function TravellingSalesman(graph) {
  const numNodes = graph.length;
  const visited = Array(numNodes).fill(false);
  visited[0] = true;
  let minCost = Infinity;
  let optimalPath = [];

  function Traverse(currentNode, path, cost, count) {
    if (count === numNodes && graph[currentNode][0] !== 0) {
      cost += graph[currentNode][0];
      if (cost < minCost) {
        minCost = cost;
        optimalPath = path.slice();
        optimalPath.push(1);
      }
      return;
    }

    for (let nextNode = 0; nextNode < numNodes; nextNode++) {
      if (!visited[nextNode] && graph[currentNode][nextNode] !== 0) {
        visited[nextNode] = true;
        path.push(nextNode + 1);
        Traverse(nextNode,
          path,
          cost + graph[currentNode][nextNode],
          count + 1
        );
```

```
        path.pop();
        visited[nextNode] = false;
      }
    }
  }

  Traverse(0, [1], 0, 1);
  return { minCost, optimalPath };
}

const graph = [
  [0, 10, 15, 20],
  [5, 0, 25, 10],
  [15, 30, 0, 5],
  [15, 10, 20, 0],
];

const result = TravellingSalesman(graph);
console.log("Minimum cost:", result.minCost);
console.log("Optimal path:", result.optimalPath.join(" -> "));
```

Input/Output:

PROBLEMS OUTPUT

```
[Running] node "e:\MEC\7th Semester\AI\LAB\travelingSalesman.js"
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 a searching algorithm used for traversing a weighted tree or graph to find the lowest-cost path from a starting node to a goal node. Unlike graphs, a tree does not contain cycles or back edges, which simplifies the UCS algorithm.

Code:

```
function uniformCostSearch(tree, goalNode) {
  const priorityQueue = new PriorityQueue();
  priorityQueue.enqueue({ node: tree, path: [], cost: 0 });

  while (!priorityQueue.isEmpty()) {
    const { node, path, cost } = priorityQueue.dequeue();
    if (node.value === goalNode) {
      return { cost, path: path.concat(node.value) };
    }
    const children = node.children;
    for (const child of children) {
      const costToChild = child.cost;
      const totalCost = cost + costToChild;
      priorityQueue.enqueue({
        node: child,
        path: path.concat(node.value),
        cost: totalCost,
      });
    }
  }
  return null;
}

class PriorityQueue {
  constructor() {
    this.elements = [];
  }
  enqueue(element) {
    this.elements.push(element);
    this.elements.sort((a, b) => a.cost - b.cost);
  }
```

```
  dequeue() {
    return this.elements.shift();
  }
  isEmpty() {
    return this.elements.length === 0;
  }
}

const tree = {
  value: "A",cost: 0,children: [
    {
      value: "B",cost: 3,children: [
        {
          value: "D",cost: 6,children: [],
        },
      ],
    },
    {
      value: "C",cost: 5,children: [
        {
          value: "E",cost: 2,children: [],
        },
        {
          value: "F",cost: 4,children: [],
        },
      ],
    },
  ],
};

const goalNode = "D";
const result = uniformCostSearch(tree, goalNode);

if (result !== null) {
  console.log("Shortest Path:", result.path.join(" -> "));
  console.log("Total Cost:", result.cost);
} else {
  console.log("Goal not reachable.");
}
```

Input/Output:

```
PROBLEMS    OUTPUT

[Running] node "e:\MEC\7th Semester\AI\LAB\ucs.js"
Shortest Path: A -> B -> D
Total Cost: 9
```