TEAM NAME: GIT TO MARS (24)
PROBLEM NUMBER: 3

1. Introduction

The 3D Traveling Salesman Problem (TSP) is a specialized variant of the classical TSP, designed to compute the most efficient travel route in a three-dimensional space. The objective is to determine the shortest possible route that allows a traveler—such as a drone, UAV, or spacecraft—to visit a series of specified waypoints exactly once and return to the starting point. The focus of this project is on fuel optimization, which directly translates to energy efficiency and cost reduction in real-world applications involving navigation in 3D environments.

This implementation is highly relevant in domains like aerospace logistics, autonomous robotic navigation, space mission path planning, and military surveillance, where navigating through 3D coordinates accurately and efficiently can significantly impact performance and sustainability. The waypoints are provided as part of a dataset that defines their unique identifiers and 3D positions. The system uses a dynamic programming approach to solve the TSP using the Held-Karp algorithm, which dramatically reduces computational time while ensuring exact solutions for moderate-sized problems.

---

2. Methodology

The Held-Karp algorithm is a well-known dynamic programming approach to solve the TSP exactly. It is particularly advantageous for cases with up to 15 cities (or nodes) due to its $O(n^2 \cdot 2^n)$ time complexity, which offers a good balance between performance and accuracy for small to medium-sized datasets.

Algorithm Overview:

Graph Construction: Each waypoint is treated as a node in a fully connected graph, with edges representing the Euclidean distance between waypoints.

State Representation: A bitmask is used to denote the set of visited nodes, enabling efficient subset management.

Dynamic Table: A dictionary dp[(bitmask, node)] stores the minimum cost of reaching a node after visiting a specific subset.

Parent Tracking: A parent dictionary is maintained to enable path reconstruction after completing the computation.

What is the Held-Karp Algorithm?

- Held-Karp is a Dynamic Programming solution to the Traveling Salesman Problem (TSP).
- Time complexity: $O(n^2 \times 2^n)$ → suitable for up to ~20 waypoints.
- It finds the shortest possible route that:
    - Visits each waypoint exactly once,
    - Returns to the starting waypoint,
    - Minimizes total travel distance (fuel).

Why Held-Karp?

Feature	**Reason**
Exact Algorithm	Guaranteed to find the optimal path (not just approximation).
Efficient for N ≤ 15	Works within the constraints of the problem (5–15 waypoints).
No Repetitions	Automatically prevents revisiting the same point.
Deterministic	Same input gives the same output — perfect for testing and validation.

How It Works – Step-by-Step

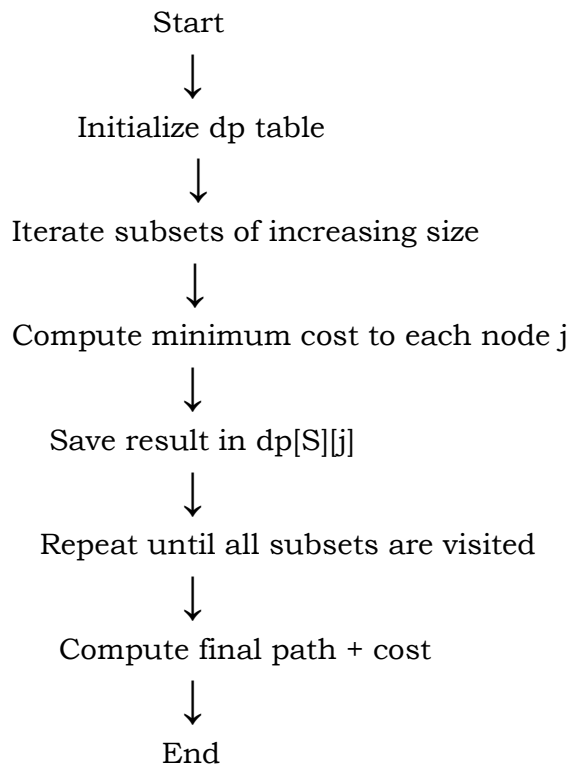
1. Input: A distance matrix `dist[i][j]` for N waypoints.
2. Define a DP table:
   `dp[S][j]` = minimum cost to reach subset S ending at node j.
3. Base Case:
   `dp[{0}][0] = 0` → cost to start at node 0 is 0.
4. Recurrence:
   `dp[S][j] = min(dp[S - {j}][k] + dist[k][j]) for all k in S - {j}`

5. Final Answer:
   min(dp[all nodes][j] + dist[j][0]) for all j ≠ 0

Pseudocode

```
# Held-Karp Algorithm

for all subsets S of {1, ..., n-1}:

    for all j in S:

      dp[S][j] = min(dp[S - {j}][k] + dist[k][j]) for all k in S - {j}

# Final result:

min(dp[all_nodes][j] + dist[j][0]) for j ≠ 0
```

Flowchart Summary



How to Run the Code

1. **Project Setup**:
    - Ensure all HTML, CSS, and JS files are placed in the same directory.
    - Include this JS file via a <script src="yourFile.js"></script> tag in your HTML.
2. **Steps**:
    - Open the HTML file in a web browser (no need for Python or servers unless you're using server-side logic).
    - Use the **file input** to upload a waypoints.txt file.
    - Enter a valid **Start Node ID** in the input field.
    - Click "**Process**" to visualize the TSP route and get results.
    - Click "**Download Path**" to save the output as path.txt.

Dataset Handling: Parsing **waypoints.txt**

Format of waypoints.txt:

```
# ID   X     Y     Z
1     10.2  15.3  5.0
2     20.1  25.3  15.0
...
```

- **Parsing Logic** (parseWaypoints function):
    - Each non-empty line (ignoring comments starting with #) is split by whitespace.
    - Expects **4 values per line**: id x y z.
    - The values are parsed and pushed into the waypoints array as: { id: 1, x: 10.2, y: 15.3, z: 5.0 }

Output Format: path.txt

Structure:
<id1> <id2> <id3> ... <idN> <totalFuelCost>
Example:
1 3 5 2 1 124.65

- **Meaning**:
    - A complete TSP path starting and ending at the same node.
    - The last number is the **total fuel cost** (total distance covered in 3D space).

Challenges and Solutions

1. Computational Complexity (TSP is NP-hard):

- **Challenge**: Exhaustive search is infeasible beyond ~10 nodes.
- **Solution**: Implemented **Dynamic Programming with Bitmasking** (Held-Karp algorithm) which works efficiently for up to **15 nodes**.
    - Time complexity: O(n^2 * 2^n) where n = number of waypoints.
    - Memory-efficient memoization using memo[pos][visited].

2. 3D Distance Calculation:

- **Challenge**: Euclidean distance had to be adapted for 3D coordinates.
- **Solution**: Used:

  sqrt((x2 - x1)² + (y2 - y1)² + (z2 - z1)²)

3. Waypoint ID vs Array Index:

- **Challenge**: User inputs ID, but arrays use index-based logic.
- **Solution**: Used findIndex() to map ID to internal index: waypoints.findIndex(wp => wp.id === startNodeId)

8. Future Improvements

- **Scalability Enhancements:** Implement Simulated Annealing or Genetic Algorithms to tackle datasets with more than 20 waypoints.
- **Parallel Computation:** Use Python's multiprocessing or concurrent.futures to parallelize subset evaluations and distance matrix calculations.
- **Visualization:** Integrate matplotlib, Plotly, or Mayavi to create interactive 3D plots of routes.
- **User Configuration:** Add support for user-specified constraints via a configuration file (e.g., config.json) or command-line arguments.
- **Real-Time Integration:** Link the system with real-time location feeds for dynamic re-planning of routes in response to new waypoints or obstacles.

9. References

- https://docs.python.org/3/library/math
- https://compgeek.co.in/held-karp-algorithm-for-tsp/#google_vignette
- https://github.com/arshad-muhammad/kvgce-hackwise.git