**Overview of Task3\_2.ipynb**

The Task3\_2 notebook focuses on solving the problem of aligning raw GPS trajectory data with a road network, commonly referred to as map matching. The objective is to accurately map GPS trajectories onto road segments while addressing challenges such as GPS noise, sparse data points, and complex trajectories involving turns, gaps, or inconsistencies. The approach is structured into four key steps: **data cleaning**, **spatial indexing**, **map matching using probabilistic algorithms**, and **result analysis and visualization**. The solution integrates mathematical and computational techniques, such as distance metrics, graph theory, and dynamic programming (e.g., the Viterbi algorithm), to process and analyze real-world GPS data effectively.

**Overall Approach and Key Features**

The solution begins with **data cleaning**, which removes noise and outliers from raw GPS data, ensuring reliability and spatial consistency. Next, it employs **spatial indexing** techniques, creating efficient structures like KDTrees for fast querying of nearby road segments. The core of the solution is the **map matching algorithm**, which matches GPS points to road segments by leveraging probabilistic models, considering factors like spatial proximity, road connectivity, and trajectory continuity. Finally, it provides **tools for analysis and visualization**, enabling users to evaluate the matched trajectories and interpret the results.

Key features include:

* **Enhanced Trajectory Cleaning**: Removes outliers and improves data quality using statistical methods like Local Outlier Factor (LOF).
* **Advanced Spatial Indexing**: Employs dual KDTree structures for efficient and precise nearest-neighbor searches.
* **Improved Map Matching Algorithm**: Handles trajectory complexities such as turns, gaps, and noise by considering road properties like bearings and connectivity.
* **Visualization Tools**: Provides detailed plots showing the original and matched trajectories for easy interpretation.
* **Analysis Tools**: Offers functionality to compute statistics and evaluate the accuracy of the matched trajectories.

**Detailed Explanation of Each Component**

**1. EnhancedTrajectoryCleaner Class**

The purpose of this class is to preprocess raw GPS trajectories by detecting and removing outliers. It ensures that the cleaned trajectories are spatially consistent and suitable for map matching.

* **Key Concepts**:
  + **Haversine Distance**: Calculates the great-circle distance between two points on a sphere, making it essential for identifying unrealistic jumps in trajectories.
  + **Local Outlier Factor (LOF)**: Detects GPS points that deviate significantly from their neighbors, indicating potential noise or errors.
  + **Temporal Consistency**: Ensures that the removal of outliers does not disrupt the logical flow of the trajectory.
* **Key Methods**:
  + \_haversine\_distance: Implements the formula:

Where:

* d: The distance between the two points (along the surface of the sphere) in the same unit as the radius R.
* R: The radius of the sphere (Earth's radius is approximately **6,371 km** or **3,959 miles**).
* ϕ1,ϕ2 ​: Latitudes of the two points (in radians).
* λ1,λ2 ​: Longitudes of the two points (in radians).
* Δϕ: Difference in latitudes (ϕ2−ϕ1 ​).
* Δλ: Difference in longitudes (λ2−λ1).
  + \_mark\_outliers: Uses LOF to dynamically detect points that deviate from the local trajectory pattern.
  + \_clean\_trajectory: Applies outlier detection and removes identified points while retaining a minimum number of trajectory points.
* **Idea**: By applying statistical and spatial methods, this class ensures that the cleaned data is accurate and reliable for further processing.

**2. AdvancedSpatialIndex Class**

This class builds efficient data structures for finding the nearest road segments to a GPS point, improving computational performance for large datasets.

* **Key Concepts**:
  + **KDTree**: A spatial data structure used for fast nearest-neighbor searches.
  + **Interpolation**: Breaks road segments into smaller sub-segments to handle complex curves and improve matching accuracy.
* **Key Methods**:
  + \_build\_index: Constructs two KDTrees—one for road endpoints and another for interpolated points along road segments. This dual indexing strategy ensures both precision and speed.
  + nearest\_edges: Combines KDTree queries and precise distance calculations to identify the closest road segments to a GPS point.
  + \_distance\_to\_segment: Computes the shortest distance from a point to a line segment using vector projections.
* **Idea**: Spatial indexing enhances the scalability and efficiency of the algorithm, enabling it to handle large road networks with high precision.

**3. ImprovedMapMatcher Class**

This class implements the map matching algorithm, assigning GPS points to road segments based on probabilistic models and logical continuity.

* **Key Concepts**:
  + **Emission Probability**: Measures how likely a GPS point belongs to a road segment based on proximity.
  + **Transition Probability**: Evaluates the likelihood of moving from one road segment to another, considering angle differences, connectivity, and road topology.
  + **Viterbi Algorithm**: A dynamic programming approach to find the most probable sequence of road segments corresponding to the trajectory.
* **Key Methods**:
  + \_find\_candidate\_edges: Identifies road segments near a GPS point and filters candidates based on connectivity and angle consistency.
  + \_calculate\_transition\_probability: Combines geometric (angle-based) and topological (connectivity-based) factors to compute transition probabilities.
  + \_viterbi\_matching: Uses the Viterbi algorithm to maximize the overall probability of a trajectory alignment.
  + match\_trajectory: Cleans the trajectory, identifies candidate edges for each point, and applies the Viterbi algorithm to map the trajectory to the road network.
* **Idea**: This class ensures that GPS trajectories are mapped to the road network in a way that balances spatial accuracy and logical consistency.

**4. Utility Functions**

These functions provide additional support for data input, analysis, and visualization, making the code user-friendly and interpretable.

* **Key Functions**:
  + process\_trajectory: Parses raw trajectory strings and converts them into usable coordinate formats.
  + map\_match\_trajectories: Processes multiple trajectories and applies the map matching algorithm, collecting results for analysis.
  + plot\_match\_result and visualize\_matches: Generate visualizations of matched trajectories, highlighting original points, cleaned trajectories, and matched road segments.
  + analyze\_results: Computes statistics such as the number of matched edges and the average confidence scores.
  + load\_network\_and\_data: Loads road network data from OpenStreetMap and GPS trajectory data from CSV files.
  + plot\_single\_trajectory\_improved: Provides enhanced visualizations of individual trajectories, including trajectory statistics and matched segments.
* **Idea**: These utility functions simplify data handling and enable clear visual and statistical analysis of the map matching results.

**Algorithms and Logic**

1. **Outlier Detection and Trajectory Cleaning**: Combines statistical methods (LOF) with trajectory-specific heuristics (speed, turn angles, and local density) to identify and remove unreliable points, ensuring temporal consistency.
2. **Spatial Indexing**: Uses KDTree structures to perform efficient nearest-neighbor searches for road segments, handling large datasets with high accuracy.
3. **Map Matching**:
   * **Emission Probability**: Assesses the likelihood of a GPS point aligning with a road segment based on distance.
   * **Transition Probability**: Accounts for logical connections between road segments, penalizing unrealistic transitions.
   * **Viterbi Algorithm**: Finds the most probable sequence of road segments corresponding to the trajectory.
4. **Visualization**: Combines original GPS points, cleaned trajectories, and matched road segments into clear and informative plots using Matplotlib, enhancing interpretability and debugging.

END