Dynamic Programming Shortest Path Finder

name: Su Hyun Kim
SJSU ID: 018219422
class #: CMPE-252 Sec 02
source code link: github link

Introduction

This project implements a Dynamic Programming (DP) approach to find the shortest path between two nodes in a graph. The script reads graph data from input files, processes the graph, and computes the optimal paths from a start node to an end node. Results include the shortest path, cumulative distances to each node, and execution time.

Implementation Overview

Language and Tools

- **Programming Language**: Python 3.10
- **Libraries Used**:
  - `collections` for data handling
  - `tqdm` for progress tracking

Files Structure

- **Main Script**: `018219422.py`
- **Input Files**:
  - `input.txt`: Contains graph edges and nodes
  - `coords.txt`: Contains coordinates for each node
- **Output Files**:
  - `<SJSU_ID>.txt`: Contains the shortest path and distances

Algorithm Implementation

Data Processing

- **Graph Representation**
  - The graph is represented as an adjacency list using a dictionary (`edges_child_info`) where each key is a node, and each value is a list of tuples representing connected nodes and the distances to them.
- **Node Information**
  - Node coordinates are stored in `node_info`, a list indexed by node number.

Dynamic Programming Approach

- **Initialization**: Sets up a memoization list `dp` to store the minimum cumulative distance (cost-to-come) and path for each node. The `dp` entry for the start node is initialized to zero.
- **Path Calculation**: Iteratively updates the `dp` table using a deque to process nodes. For each node, the algorithm calculates the minimum cumulative distance to its neighbors and updates paths accordingly.

Key Components

- **Distance Tracking**: The `dp` list keeps the shortest cumulative distance and path for each node.
- **Path Reconstruction**: For each node, the `dp` entry holds the shortest path from the start node, reconstructed at the end from the path entries.

Output and Execution

Output

The results are saved in `<SJSU_ID>.txt`:

- **Optimal Path**: The first line lists the shortest path from the start to the end node.
- **Cumulative Distances**: The second line lists the cumulative distances to each node from the start.

Execution

The total execution time is recorded and displayed in the logs.

Instructions

Prerequisites

- **Python Version**: 3.10 or higher
- **Libraries**: `tqdm`

Setup

1. **Virtual Environment** (optional but recommended)
   ```
   python -m venv venv
   ```

Activate the environment

- **Windows**: `venv\Scripts\activate`
- **macOS/Linux**: `source venv/bin/activate`

2. Install Dependencies
   `pip install -r requirements.txt`

Preparing Input Files

Ensure `input.txt` and `coords.txt` are in the same directory as `018219422.py`.

Running the Script

Run the script to calculate the shortest path and generate output:

```
python 018219422.py
```

Code Structure

DataProcessor Class

- **Initialization**
  - Sets up graph and node information.
- **Methods**
  - `process_input_files`: Reads and processes graph and coordinate input files.
  - `dynamic_programming`: Computes the shortest path using a DP approach with memoization.
  - `generate_output_file`: Writes the optimal path and distances to the output file.
  - `main`: Orchestrates the execution of all methods.

Results

The output shows the shortest path and cumulative distances from the start node to each node. The execution time is also recorded, demonstrating the efficiency of the DP approach for this problem.
