Dijkstra's Algorithm Visualization

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

Dijkstra's Algorithm Visualization **1**
- Introduction 1
- Implementation Overview 2
  - Language and Tools 2
  - Files Structure 2
- Algorithm Implementation 2
  - Data Processing 2
  - Dijkstra's Algorithm 3
  - Visualization 3
  - Video Generation 3
- Instructions to Compile and Run the Code 3
  - Prerequisites 3
  - Setting Up the Environment 4
    - Using requirements.txt 4
    - Using Poetry (Optional) 4
  - Preparing Input Files 4
  - Running the Script 4
    - Command-Line Arguments 4
    - Examples 5
- Code Explanation 5
  - Structure 5
  - Key Components 5
- Conclusion 6
- Future Enhancements 6

Introduction

This report details the implementation of Dijkstra's algorithm and visualization for finding the shortest path between two nodes in a graph. The visualization component generates images of each step of the algorithm and can optionally create a video of the process. The output includes the shortest path, distances, and visual representations.

Implementation Overview

Language and Tools

- **Programming Language**: Python 3.10
- **Libraries Used**:
  - `matplotlib` for visualization
  - `opencv-python` for video generation
  - `argparse` for command-line argument parsing
  - `collections`, `heapq`, and `os` for data handling

Files Structure

- **Main Script**: `main.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
  - `<SJSU_ID>.mp4`: Video of the algorithm's execution
  - `images/`: Directory containing images generated during the algorithm

---

Algorithm Implementation

Data Processing

- **Graph Representation**:
  - The graph is represented using a dictionary (`defaultdict(list)`) where each key is a node, and the value is a list of tuples representing connected nodes (neighbors) and the distance to them.

- **Node Information**:
    - Node coordinates are stored in a list (`node_info`), where the index corresponds to the node number. (0th index have no meaning)

Dijkstra's Algorithm

- **Priority Queue**:
    - Uses a min-heap (`heapq`) as a priority queue to always select the node with the smallest cumulative distance.
- **Distance Tracking**:
    - Maintains a dictionary (`min_distance_from_start`) to keep track of the minimum distance to each node from the start node.
- **Path Reconstruction**:
    - Keeps track of the path and distance history to reconstruct the shortest path at the end of the algorithm.

Visualization

- **Image Generation**:
    - Generates images at each significant step of the algorithm, showing the nodes being processed.
    - Uses Matplotlib to plot the graph, nodes, and edges.
    - Nodes are color-coded:
        - **Green**: Start node
        - **Red**: End node
        - **Blue**: Nodes currently being processed
        - **Gray**: Nodes that have been fully processed
- **Final Path Visualization**:
    - After finding the shortest path, generate a final image highlighting the path in red.

Video Generation

- **Using OpenCV**:
    - Reads the generated images and compiles them into a video.
    - Video is saved in MP4 format using the `mp4v` codec.
- **Command-Line Control**:
    - Video generation can be toggled on or off using command-line arguments.

Instructions to Compile and Run the Code

Prerequisites

- **Python 3.10** or higher (Python **3.10.5** preferred)
- **FFmpeg** installed and accessible via the system's PATH
- **Required Python Libraries**:
    - `matplotlib`
    - `opencv-python`
    - `numpy` (dependency of `matplotlib` and `opencv-python`)

Setting Up the Environment

Using `requirements.txt`

1. **Create a Virtual Environment** (optional but recommended):
   `python -m venv venv`
2. **Activate the Virtual Environment**:
    - **Windows**: `venv\Scripts\activate`
    - **macOS/Linux**: `source venv/bin/activate`
3. **Install Dependencies**: `pip install -r requirements.txt`

Using Poetry (Optional)

1. **Install Poetry**: `pip install poetry`
2. **Install Dependencies**: `poetry install`
3. **Activate the Poetry Shell**: `poetry shell`

Preparing Input Files

Ensure that `input.txt` and `coords.txt` are placed in the same directory as `main.py`.

Running the Script

Open a terminal in the directory containing `main.py` and the input files.

Command-Line Arguments

- **`--video`**: Control video generation. (default = 1)
    - `1` to generate the video (default).
    - `0` to skip video generation.
- **`--steps_per_frame`**: Adjust the number of algorithm steps included in each video frame. (default = 3)
    - Higher values speed up the video by including more steps per frame.

Examples

- `python main.py --video 1 --steps_per_frame 5`

Code Explanation

Structure

- `DataProcessor` **Class**:
  - **Initialization**:
    - Accepts parameters to control video generation and steps per frame.
    - Initializes variables for graph and node information.
  - **Methods**:
    - `process_input_files`: Reads and processes input files.
    - `dijkstras_algorithm`: Implements Dijkstra's algorithm with visualization steps.
    - `generate_base_graph_image`: Creates the base graph image used in visualizations.
    - `generate_image_for_step`: Generates images at each step of the algorithm.
    - `generate_final_image`: Highlights the shortest path on the graph.
    - `generate_output_file`: Writes the shortest path and distances to a text file.
    - `generate_video_from_images`: Compiles images into a video using OpenCV.
    - `main`: Orchestrates the execution of all methods.
- **Command-Line Argument Parsing**:
  - Uses `argparse` to handle command-line arguments for video generation and frame rate control.

Key Components

- **Visualization Logic**:
  - Nodes are plotted using Matplotlib.
  - Colors indicate the state of nodes during the algorithm.
  - Images are saved at specified intervals based on `steps_per_frame`.
- **Algorithm Logic**:
  - Uses a priority queue to select the next node with the smallest cumulative distance.
  - Keep track of the minimum distance to each node to avoid unnecessary processing.

Conclusion

The implemented script successfully calculates the shortest path using Dijkstra's algorithm and provides visual insights into the algorithm's execution.

Future Enhancements

- **Performance Optimization**:
    - Explore ways to optimize the visualization process to handle larger graphs efficiently.
    - For example, we can make graph images and save directly to memory (in Python, use BytesIO), and this will save time.
