Spring 2021
Introduction to Artificial Intelligence

Homework 2: Route Finding

Due Date: 2021/4/13 23:55

Introduction

The goal of this programming assignment is to implement a variety of search algorithms you learned in class. You are given real data in Hsinchu exported from OpenStreetMap. Given a starting point and destination, different search algorithms find different solutions. In this assignment, you will see the difference displayed on an actual map.

Setup

Additional packages are required to visualize your results on a web-based interactive map. Please follow the following installation instructions.

For local development:
Run the command to install packages

```
pip install folium
pip install jupyter
```

or

```
conda install folium -c conda-forge
conda install jupyter
```

After installation, you can execute “jupyter notebook” in your terminal. Then open main.ipynb to run the test.

For Google Colab:
Google Colab is based on Jupyter Notebook. You can open main.ipynb directly. Then run the first cell to install the required package.

```
!pip install folium
```

Data

To model the route finding as a search problem, we leverage the state-based model and represent a map with intersections. In OpenStreetMap, each intersection is labeled with a unique ID. A road connects two nodes. All roads in North and East District, Hsinchu City are in edges.csv.

The CSV file stores the following information:

column	detail
start	The ID of the starting node of a road.
end	The ID of the end node of a road.
distance	The length of a road. (Unit: meter)
speed limit	The speed limit of a road. (Unit: km/h)

For A*, we use straight-line distance as the heuristic function. Therefore, we provide the information in heuristic.csv.
The detail about column is following:

column	detail
node	The ID in edge.csv
ID1	The straight-line distance from node to ID1. (Unit: meter)
ID2	The straight-line distance from node to ID2. (Unit: meter)
ID3	The straight-line distance from node to ID3. (Unit: meter)

The file graph.pkl is graph information for drawing your path. You do not have to deal with it. Note that, please make sure graph.pkl and main.ipynb are in the same folder.

Requirements

The code we provided is used to display your results on a map. You can implement search algorithms in main.ipynb directly or in different new .py files. Each function should read the data we provide. You can store data in any data structure. However, the standard Python library **is only allowed** in this assignment.

Part 1: Breadth-first Search (10%)

- Write a breadth-first search function to find a path from a starting node to an end node.
- The function detail is as follow:

name	bfs	
parameters	start	Type: integer The starting node ID
	end	Type: integer The end node ID
returns	path	Type: list of integer The path you found, stored as a list of node IDs. The first is starting node ID. The last is end node ID.
	dist	Type: float The distance of the path you found. (Unit: meter)
	num_visited	Type: integer The number of nodes were visited when you search.

Part 2: Depth-first Search (10%)

- Write a depth-first search function to find a path from a starting node to an end node.
- You can implement depth-first search in a recursive method or a non-recursive method.
- The function detail is as follow:

name	dfs	
parameters	start	Type: integer The starting node ID
	end	Type: integer The end node ID
returns	path	Type: list of integer The path you found, stored as a list of node IDs. The first is start node ID. The last is end node ID.
	dist	Type: float The distance of the path you found. (Unit: meter)
	num_visited	Type: integer The number of nodes were visited when you search.

Part 3: Uniform Cost Search (20%)

- Write a uniform cost search function to find the shortest path from a starting node to an end node.
- The function detail is as follow:

name	ucs	
parameters	start	Type: integer The starting node ID
	end	Type: integer The end node ID
returns	path	Type: list of integer The path you found, stored as a list of node IDs. The first is starting node ID. The last is end node ID.
	dist	Type: float The distance of the path you found. (Unit: meter)
	num_visited	Type: integer The number of nodes were visited when you search.

Part 4: A* Search (20%)

- Write a A* search function to find the shortest path from a starting node to an end node.
- The function detail is as follow:

name	astar	
parameters	start	Type: integer The starting node ID
	end	Type: integer The end node ID
returns	path	Type: list of integer The path you found, stored as a list of node IDs. The first is start node ID. The last is end node ID.
	dist	Type: float The distance of the path you found. (Unit: meter)
	num_visited	Type: integer The number of nodes were visited when you

		search.

Part 5: Test your implementation (15%)

- Compare different search algorithms on the following three test cases.
- The starting nodes and end nodes are as follow:

	starting node	end node
1	National Yang Ming Chiao Tung University (ID: 2270143902)	Big City Shopping Mall (ID: 1079387396)
2	Hsinchu Zoo (ID: 426882161)	COSTCO Hsinchu Store (ID: 1737223506)
3	National Experimental High School At Hsinchu Science Park (ID: 1718165260)	Nanliao Fighing Port (ID: 8513026827)

- In main.ipynb, please change start and end. Then run those test cells to show the results.

Part 6: Search with a different objective (Bonus) (10%)

- Write a A* search function to search the fastest path from a starting node to an end node using.
- For this problem, you can assume drivers never drive faster than the speed limit. You can calculate the new cost for edges using the speed limit. You need to consider an admissible heuristic function for this objective.
- The function detail is as follow:

name	astar_time	
parameters	start	Type: integer The starting node ID
	end	Type: integer The end node ID
returns	path	Type: list of integer The path you found, stored as a list of node IDs. The first is start node ID. The last is end node ID.
	time	Type: float The time of the path you found. (Unit: second)
	num_visited	Type: integer The number of nodes were visited when you

		search.

- Try three paths in Part 5 and compare results with Part 4.

Report (25%)

- A written report is required.
- The report should be written in **English**.
- Save the report as a **.pdf** file.
  - font size: 12
- For part 1 ~ 4, please take a screenshot of your code and explain your implementation **in detail**.
- For part 5, please take a screenshot of the results and discuss it.
- For part 6 (bonus), please take a screenshot of your code and explain your implementation **in detail**. And take a screenshot of the results and discuss it.
- Describe problems you meet and how you solve them.

Submission

Please prepare your source code and report (.pdf) into STUDENTID_hw2.zip.

e.g. 309123456_hw2.zip

Late Submission Policy

20% off per late day