Spring 2024
Introduction to Artificial Intelligence

Homework 2: Route Finding

Due Date: 4/8 (Mon.) 23:59

Introduction

When you are invited to a new restaurant located somewhere you're not familiar with, what would you do? Map applications in your smartphones have the feature that helps you plan a path from home to the restaurant. For example, Google Maps automatically figures out feasible routes and provides the corresponding instructions to reach the destination given your current position.

新竹市
恭寮
Big City遠東
巨城購物中心
二十張犁
溪洲
千甲車站
竹都城隍廟
15 分
5.8 公里
馬偕紀念醫院新竹院區
新竹
偉功里
11 分
4.8 公里
國立清華大學
將軍村停車場
好市多
30 分
東區
國立陽明交通大學
瑞昱半導體
股份有限公司
新竹科學工業園區
普天宮
台積電七廠
新竹科學園區

The abovementioned application is a **navigation system**. A navigation system involves three steps**.** First, you have to construct a map (also called **mapping**). Do you know how to build one (e.g., Hsinchu City)? Second, given a map, you have to know your current position. This step is called **localization.** Do you know how Google Maps locates your position? Third, a **route finding algorithm** that takes as input a current position and destination set by you is utilized to identify feasible routes. In this homework, we focus on the third part. Before reading the following description, how would you tackle the route finding problem?

The goal of this programming assignment is to implement a variety of **search algorithms** covered in the class. You are given real map data of Hsinchu City exported from OpenStreetMap (https://www.openstreetmap.org/). Given a starting point and destination, different search algorithms find you different routes. You will see the difference shown on an actual map.

Notes:

- Please read the **Appendix** carefully.
- Only the standard Python library **is allowed** in this assignment.
- You will implement each search algorithm in the corresponding Python script (e.g., bfs.py).

Implementation (70%)

Part 1: Breadth-first Search (10%)

- Implement a breadth-first search function to find a path from a starting node to an end node.
- The detail of the function 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 visited when you search.

Part 2: Depth-first Search (10%)

- Implement 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 detail of the function 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 visited when you search.

Part 3: Uniform Cost Search (20%)

- Implement a uniform cost search function to find the shortest path from a starting node to an end node.
- The detail of the function 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 visited when you search.

Part 4: A* Search (20%)

- Implement a A* search function to find the shortest path from a starting node to an end node.
- The detail of the function 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 visited when you search.

Part 5: Test your implementation (10%)

- 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.
- **We have provided example results in HW2_Example_Results.pdf on E3, please note that:**
    - **In BFS, the "number of nodes in the path" in your solution should match the provided results**
    - **In UCS and A*, the "distance of path" in your solution should match the provided results**

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

- Implement a A* search function to search the fastest path from a starting node to an end node using.
- For this part, you can assume that 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.
- Feel free to design your heuristic function, however, please ensure that it remains an admissible heuristic function.
- The starting nodes and end nodes are 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 visited when you search.

- Test your implementation with the three cases in Part 5
- Compare the results with results obtained in Part 4.
- **In this part, execution time of test case 3 has to be less in 1000 seconds, or the bonus points will be 0.**

Report (30%)

- A report is required.
- The report should be written in **English**.
- Please save the report as a **“report.pdf”**.
- 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 encounter and how you solve them.
- Answer the questions in the report template.

QA Page

If you have any questions about this homework, please ask them on the following Notion page. We will answer them as soon as possible. Additionally, we encourage you to answer other students' questions if you can.
https://lopsided-soursop-bec.notion.site/HW2-QA-Sheet-9a731b94bd98480d83ba4c7bbe170555?pvs=4

Submission

Due Date: 4/8 (Mon.) 23:59

Please directory compress all your code files and report (.pdf) into {STUDENT ID}_hw2.zip (ex. 109123456_hw2.zip) and **submit it to the New E3 System.**
The file structure should look like:

```
📂{student_id}_hw2.zip
 ├ 📜astar.py
 ├ 📜astar_time.py
 ├ 📜bfs.py
 ├ 📜dfs_recursive.py (dfs_stack.py)
 ├ 📜main.ipynb
 ├ 📜report.pdf
 └ 📜ucs.py
```

Wrong submission format leads to -10 point

Late submission leads to -20 points per day

Appendix

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 formulate the route finding problem 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.