

Spring 2024

Introduction to Artificial Intelligence

Homework 1: Route Finding

Due Date: 03/21 16:59

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.



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 (60%)

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.

- You need to provide the screenshot of your three results in Part 5 to the report. The pdf file should contain the screenshots photos only.
- 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.
- Read the heuristic values from `heuristic_values.csv`. Do not make heuristic values yourselves.
- 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

- For Part 1~3, you need to 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 Fighting Port (ID: 8513026827)

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

- Scoring will be based on 10 test cases, including the three mentioned above.
- Each part's final score (except for DFS) will be calculated by multiplying its total score by the accuracy of the 10 test cases.
- For DFS, you will receive the score as long as you can reach the destination in all three examples.
- **We have provided example results in HW1_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

Demo (40%)

- In this part, the TA in charge will ask you some questions.
- Please fill out the schedule form, which we will announce later. Please come to the demo on time.
- The demo regularization will be announced then. It would be on week 6.

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://www.notion.so/HW1-QA-4de0437e82ce442ca3c6c77de43fe3c3?pvs=4>

Submission

Due Date: 03/21 (Fri.) 16:59

Submit all your files to the New E3 System.

The file structure should look like:

```
{Student_ID}_astar.py
{Student_ID}_bfs.py
{Student_ID}_dfs_result.pdf
{Student_ID}_dfs_recursive.py / {Student_ID}_dfs_stack.py
{Student_ID}_ucs.py
```

* {Student_ID} does not include '{' and '}'

Warning

- Wrong submission format leads to -10 point
- No late submission.
- Anyone who discloses the demo questions after the demonstration will receive a zero for the entire assignment.

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)

For A*, we use straight-line distance as the heuristic function. Therefore, we provide the information in [heuristic_values.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.