ESC190 LAB 1

Robotic Path Planning

Due: Feb 9, 2020; 23:59

BACKGROUND

Robotics and art can go hand in hand. If this idea resonates with you, you may like the content of the following:

https://www.nextrembrandt.com/
https://mjhung.wixsite.com/robotart-ntu
https://paintschainer.preferred.tech/index_en.html

Top Secret AI (TSA - Toronto, Canada) has commissioned you to develop software for their robot ArtBotCapture (also known as ABC). ArtBotCapture learns how to make artwork based on its observations (3D "captures") of real-world art installations. TSA has spent millions of dollars developing this capability. At the very end, due to poor project planning, they have realized that they are out of money and have not thought about how ABC will obtain all these 3D captures on its own. ABC is fully equipped with GPS sensors, a differential wheel set, and autonomous localization and mapping. TSA has reached out to you to develop software which will plan the path of all the art installations it will visit in Toronto. You will work with data that lists the public artwork installations in Toronto (found here). Due to memory constraints, ABC cannot store a map of all the travelable roads and streets in Toronto. However, it is guaranteed that a path exists between 2 installations if they are in the same **ward** or if they are in numerically adjacent **wards**.

STARTER FILES

- From the terminal, run: **cd ~/Desktop/ESC190_Labs/<utorid>** or if you have a different name for the directory, change to that directory.
- Run **git pull.**
- Run **ls** to see the *lab1* directory.

PROVIDED DATA

- `public_artwork_data.txt`: Real data of Toronto's public artwork installations.
- `test_data.txt`: Fake data formatted for testing purposes.

PROVIDED CODE

- `lab1_utilities.py`: do **not** modify this file. The Graph and Installation Classes are defined here.
- `tester_lab1.py`: you should add all code for testing in this file. This file will not be graded.

- `lab1.py`: starter code for this lab. This is the **only** file that will be marked. Do **not** add any code outside of the provided function definitions.

TASKS

You set out to implement Dijkstra's algorithm for ABC's path planning. TSA has provided a starter script, `lab1.py`, with 5 functions for you to complete. Please read through all the function descriptions carefully and understand their relationship with each other before diving in.

`get_installations_from_file(file_name)`

Parse the data from the .txt file to create and return a list of instances of `lab1_utilities.Installation`. Each line of the file represents a single installation, except for the heading. Each `Installation` instance should have attributes `str` name, `int` ward, a 2-element `tuple(float, float)` position (x, y), and `bool` indoors (`True` for indoor, `False` for outdoor).

Input format

`file_name: str` indicating the file name with the list of art installations.

Return format

`List[lab1_utilities.Installation]`

Sample inputs

```
>>> installations = get_installations_from_file('test_data.txt')
>>> for installation in installations:
>>>     print(installation)
Installation A in Ward 2
Installation B in Ward 3
Installation C in Ward 4
Installation D in Ward 4
Installation E in Ward 6
```

`euclidean_distance(position1, position2)`

Calculates the Euclidean distance between two 2D points in Cartesian coordinates.

Input format

`position1, position2: tuple(float, float) representing (x, y)`

Return format

`float`

Sample inputs

```
>>>euclidean_distance((0, 0),(1, 1))
```

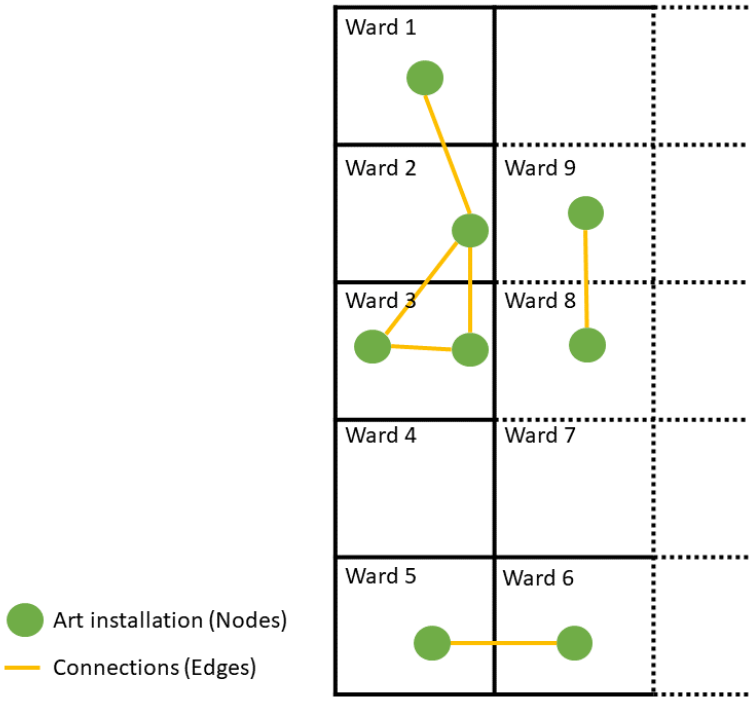
```
1.4142135623730951
```

`get_adjacency_mtx(installations)`

Create an adjacency matrix given a list of installations. The matrix will have a size of **V**x**V**, where **V** is the total number of installations. The matrix will be filled with floating-point values, where a value of zero indicates no edge.

Art installations are connected by an edge if they are in the same or numerically adjacent ward(s), as shown in the figure below. The weight of each edge is the Euclidean distance between the two art installations multiplied by a scaling factor k.

- $k = 1.0$ if **both** art installations are outdoors
- $k = 1.5$ if **one or both** art installations are indoors



Input format

`installations: List[lab1_utilities.Installation]`

Return format

`List[List[float]`

Sample inputs

```
>>> installations = get_installations_from_file('test_data.txt')
>>> get_adjacency_mtx(installations)
```

```
[[0, 1.5, 0, 0, 0],
 [1.5, 0, 1.5, 6.0, 0],
 [0, 1.5, 0, 4.123105625617661, 0],
 [0, 6.0, 4.123105625617661, 0, 0],
 [0, 0, 0, 0, 0]]
```

`make_graph(installations)`

Given a list of installations, build the undirected graph representing the dataset as defined by the class `lab1_utilities.Graph`. The `Graph.installations` attribute should be a `List[str]`, a list of the names of the installations. *Hint: use the other functions you wrote for this lab.*

Input format

`installations: List[lab1_utilities.Installation]`

Return format

`lab1_utilities.Graph`

Sample inputs

```
>>> installations = get_installations_from_file('test_data.txt')
>>> print(make_graph(installations))
['A', 'B', 'C', 'D', 'E']
[[0, 1.5, 0, 0, 0], [1.5, 0, 1.5, 6.0, 0], [0, 1.5, 0, 4.123105625617661,
0], [0, 6.0, 4.123105625617661, 0, 0], [0, 0, 0, 0, 0]]
```

`find_shortest_path(installation_A, installation_B, graph)`

Implement Djikstra's algorithm to find the shortest path between two art installations given the undirected, weighted graph. The graph will be made using `make_graph` and the two installations will be given by their name attributes as strings. The output will be the shortest path given in terms of a sequence of nodes.

Input format

`installation_A, installation_B: str` *e.g., 'CANADIAN VOLUNTEERS WAR MEMORIAL'*

`graph:` an instance of class `lab1_utilities.Graph`

Return format

`tuple(float, List[str])`

A 2-element `tuple`. The first element is a `float` representing the total travelled distance from `installation_A` to `installation_B` when taking the path identified in the second element. The

second element is a `List[str]` of the names of the installations representing the shortest path. If there is no path between the two installations, return `(None, [])`

Sample inputs

```
>>> installations = get_installations_from_file('test_data.txt')
>>> find_shortest_path("A", "D", make_graph(installations))
(7.123105625617661, ['A', 'B', 'C', 'D'])
```

SUBMISSION INSTRUCTIONS

- Remember to stage your files, commit, then push. You can and should commit frequently.
- No late submissions will be accepted. The submission closest to and before the indicated due date will be marked.
- Your code must run on Python3.6.9 on the Linux ECF machines
- DO NOT ASSUME your code will run after minor modifications – TEST IT as this is the only way to be certain.
- Review the ESC190 Handbook if you are unsure whether your submission was successful.