DEPARTMENT *of* COMPUTING

College of Business & Technology

EAST TENNESSEE STATE UNIVERSITY

CSCI 5260 – Artificial Intelligence

Project 1 – Simple and Heuristic Searches

Part 1 – The Buc Mobile

Background

The BucMobile is an ETSU-created autonomous vehicle that has access to Open Street GPS data from OpenStreetMap.org. BucMobile needs your help in using its data to navigate around the city of Johnson City. The BucMobile's starting location will always be the stop sign next to Reece Museum on ETSU's campus. The BucMobile would like you to test several routes **using four algorithms**. You will need to present your results in a series of figures with explanations.

Preparation

Necessary Libraries

1. Probably needed from pip install—
   a. `osmnx` – A library that allows you to pull data from the Open Street GPS and store it as an undirected graph of GPS coordinates.
      i. https://osmnx.readthedocs.io/en/stable/
   b. `plotly.graph_objects` – plotly is a data visualization library, and the graph_objects class represents parts of a figure.
      i. https://plotly.com/python/
   c. `networkx`
2. Built-in Python libraries—
   a. `collections` – allows you to use python's double-ended and other queue structures.
   b. `numpy` – the Python Scientific Computing package
   c. `matplotlib.pyplot` (optional) – for providing additional data plot functionality.

Provided Functions

I have provided the following two functions in the **project1_mapper.py** file on D2L.

1. `node_list_to_path(gr, node_list)`

   Given a `networkx` graph (`gr`) and an ordered list of nodes (`node_list`) in the format [osmid, osmid, osmid, …], return the list of edges (given as a list of location pairs) that follows the path defined by the nodes

2. `plot_path(lat, long, origin_point, destination_point)`

   Given (1) a path-ordered list of latitudes, (2) a path-ordered list of longitudes, (3) an origin point as a tuple in the format `(osmid, {"y": latitude, "x": longitude, "osmid": long_integer})`, and (4) a destination point as a tuple in the format `(osmid, {"y": latitude, "x": longitude, "osmid": long_integer})`, open a browser and display a map of the route.

Requirements

Create a driver program that traverses six given routes (listed below), using three different uninformed search algorithms. **You must use breadth-first search and depth-first search**. You may choose any other uninformed search algorithm for the other one.

1. Create a graph based on Open Street data from the address '1276 Gilbreath Drive, Johnson City, TN, USA' at a distance of 4000 meters, and a network_type of 'drive'. Example code is provided for the 1 km area around the White House in Washington, DC.
2. Create the origin point at 36.30321114344463 latitude and -83.36710826765649 longitude. Use the `osmx` library's `get_nearest_node` function to locate the node in the graph closest to that point.
3. Use the following routes in your program [You can look up the GPS coordinates yourselves]:
    a. Route 1: ETSU to Walmart (West Market Street)
    b. Route 2: ETSU to Target (North Roan Street)
    c. Route 3: ETSU to the Tweetsie Trail entrance (Legion Street at Alabama Street)
    d. Route 4: ETSU to Frieberg's German Restaurant (Main Street at Tipton Street)
    e. Route 5: ETSU to Food City (South Roan Street)
    f. Route 6: ETSU to Best Buy (Peoples Street)
4. Create the following algorithms, placing each in its own function, to test each route:
    a. Breadth-First Search
    b. Depth-First Search
    c. Developer's Choice of an uninformed search algorithm

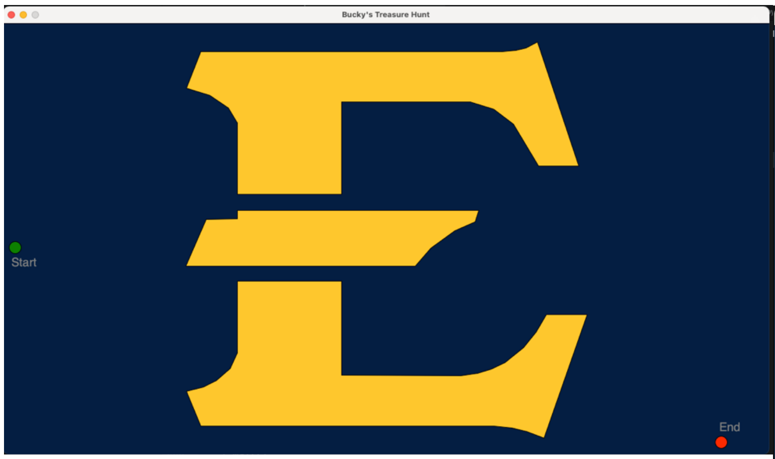
    Hint: In each function, make sure you track the nodes you visit—but also track the node you came from for that particular visit. Doing so will allow you to recreate the path after reaching the destination.
5. Suggested algorithm: create a `backtrack` function that accepts your visited list and your destination. It then works backwards to the origin so you know exactly which path you took to locate the destination. You should return a list of osmids in the order of the path (forward or backward; it doesn't matter).

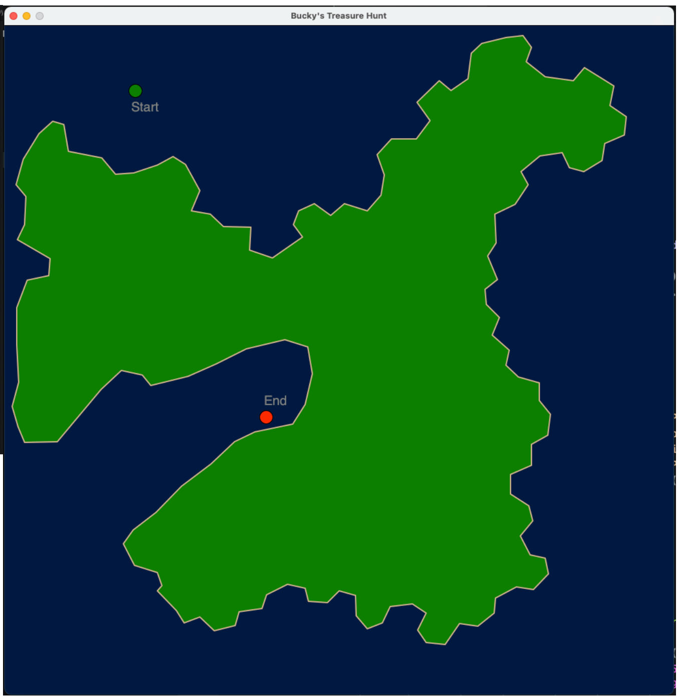
    Hint: It might also be helpful to return a list of latitudes and a list of longitudes in the same order as your route list. This can be passed into the `plot_path` function provided.
6. Use the information you have collected, and present it in a written report. You must include the following:
    a. An analysis of the search space. You may need to explore the graph a bit to do this effectively.
    b. For each route, for each algorithm, analyze
        i. The algorithm's efficiency based on runtime [use Lab 2 code as an example for collecting this data], and
        ii. The algorithm's efficiency based on steps taken. You can count a "step" however you wish, but you need to explain how you're counting in the writeup.
    c. An overall analysis of which search is best for this situation. To receive full credit, you must explain each search algorithm and why it is or is not suited for this problem.

    It would be nice to see tables and/or charts here, with brief explanations rather than wordy paragraph formats. You should incorporate images of each of your map routes.

Part 2 – Bucky's Treasure Hunt

ETSU's mascot, Bucky the Bucanneer is looking for his treasure. Bucky wants to locate it quickly to ensure that the other Bucanneers out there don't steal it before he retrieves it. Since he knows that ETSU has the most awesome Computer Science program on the Seven Seas, he wants you to simulate his treasure hunt in a computer program. Bucky will be glad to give you his two existing navigation charts, although, he also expects you to create a third navigation chart so he knows that you've tested your program thoroughly.





These images show the two maps that are available for you to practice. On the left, you must help Bucky (located at Start) navigate the ETSU logo to find his treasure (located at End). On the right is a real map of Bucanneer Island, where Bucky had buried his treasure in at the edge of Gilbreath Lagoon. His plan is to follow the coastline from the other side of the island to locate the treasure.

Provided Files

1. `graphics.py` – a graphics library that simplifies graphical programming in Python.
2. `__init__.py` – a required file that informs Python to look for included files in the current directory.
3. `project1_treasure.py` – the skeleton program, which contains the following structures:
    a. class `Field`
        i. Attributes:
            1. `self.points` – The is the set of all points you could visit
            2. `self.path` – This is the list of Point() objects that constitute the path from start to end
            3. `self.polygons` – This is the list of obstacles within the field
            4. `self.extras` – This is a Python list that holds extra stuff that you want to erase
            5. `self.width` – The width of the field on the screen
            6. `self.height` – The height of the field on the screen
            7. `self.start` – The staring location, a Point() object
            8. `self.end` – The ending location, a Point() object
            9. `self.win` – The window to display the field
        ii. Functions:
            1. `setCoords` – sets the coordinates of the window (i.e., the "viewport")
            2. `set_background` – sets the background color
            3. `add_polygon` – adds the permanent polygon to the field and adds its points to points
            4. `add_start` – adds the start location to the field
            5. `add_end` – adds the end location to the field
            6. `get_neighbors` – for a giving Point, returns a list of Points—polygon vertexes—within the Point's line of sight.
            7. `wait` – pauses the window to await a click.
            8. `close` – closes the window
            9. `reset` – resets the window by undrawing all extras (adds new start/end points if passed)
            10. `backtrack` – recreates the path located. Uses a came_from dictionary that holds the parents of each node

11. `straight_line_distance` – calculates the Euclidean distance between two points
12. **`depth_first_search`** – the uninformed search algorithm that locates the treasure
13. **`breadth_first_search`** – the uninformed search algorithm that locates the treasure
14. **`best_first_search`** – the heuristic function that locates the treasure
15. **`astar_search`** – the A* algorithm that locates the tresure

b. `setup_game_map(f)`
   i. Adds the polygon that represents Bucanneer Island.

c. `setup_logo_map(f)`
   i. Adds the polygons that represent the ETSU Logo

d. `setup_polygon_field(f)`
   i. Adds a number of polygons that you generate

e. `main`
   i. Sets up the `Field` and the search algorithms using different start and end points.

Requirements

1. Complete the following search algorithms in the Field class:
   a. **`depth_first_search`**
   b. **`breadth_first_search`**
   c. **`best_first_search`**
   d. **`astar_search`**
2. Run each of the four algorithms against the two maps that Bucky has provided, and one that you create.
   a. The ETSU Logo Map
   b. The Bucanneer Island Map
   c. A new map of polygons that you create, and that contains at least 10 polygons of random sizes.
3. Create a report of your findings, that includes:
   a. Collected Data:
      i. Screenshots of each search/map
      ii. The runtime for each algorithm [Use the Lab2 code as an example here]
      iii. The total path cost (in Euclidean Distance) for each search/map
      iv. The total number of steps for each search/map
   b. A conclusion that gives the advantages and disadvantages of using each of the four search algorithms for Bucky's Treasure Hunt.

It would be nice to see tables and/or charts, with brief explanations rather than wordy paragraph formats.

Submission and Due Date

Submit all code, zipped into two folders: project1_part1 (The Buc Mobile) and project1_part2 (Bucky's Treasure Hunt). Each folder should be self-contained in a way that allows the code to run. You can assume that I have all necessary libraries installed.

Submit the two written reports (part 1/part 2) to the dropbox separately, and place them in the appropriate Zip folder.

Project 1 is due to the D2L dropbox at or before Monday, February 8, 2020 at 11:59 PM

Rubric

A letter grade will be assigned to each of the following, and will translate to a numeric grade based on the scale in the syllabus, and averaged into an overall percentage. As a reminder, anything below a C translates to an F by University Graduate School policy. It is provided here to appropriately reflect each level.

For source code, please add comments so I can understand what is going on. Believe it or not, some student code is difficult to read. :D

	A		B			C			D		F	Zero
	A	A-	B+	B	B-	C+	C	C-	D+	D	F	0
Part 1 – The Buc Mobile												
BFS												
DFS												
Other uninformed												
Results												
Report												
Part 2 – Bucky's Treasure Hunt												
BFS												
DFS												
Best-First												
A*												
Results												
Report												