Homework #2: Search for a Restaurant!

CSCI 364 Spring 2019 Oberlin College

Due: Friday March 1 at 1:30 PM

Background

You recently purchased a brand-new autonomous car from Alset (whose motto is "The last car you'll ever need – you're all-set!"). However, after riding around for a few weeks, you've realized that its pathfinding feature seems slow and doesn't always choose the best routes to your intended destinations. Searching around online, you find message board posts suggesting they might have used Breadth First Search as their algorithm of choice. After taking CSCI 364, you know this to be a poor decision!

You have a little free time one week and decide to implement your own pathfinding algorithm to improve your car's driving abilities. Luckily, you discover Alset cars were programmed in both Python and Java, so you have the skills necessary to update your car's software.

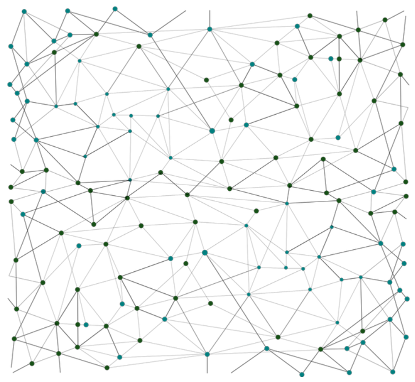
Your car comes with some predownloaded maps for you to work with, as well as some helpful classes already implemented for working with those maps:

1. `Map`, which models a graph representation of the intersections in a map and the roads that connect those intersections,
2. `MapNode`, representing an intersection in the map,
3. `Problem`, which converts the map into a search problem, providing access methods described later in the assignment instructions, and
4. `State`, representing a location in the `Problem` [only in Java; in Python these are a tuple of (latitude, longitude) pairs]

Gitting Started

To begin this assignment, please follow this link:

https://classroom.github.com/a/f2Fu0tAy



Assignment

Over the next week, you will implement three search algorithms in either Python or Java to be uploaded into your car:

1. BFS (to verify that your suspicions about their design decision are correct),
2. A* Search (to find the optimal routes quickly), and
3. Uniform Cost Search (to verify that A* finds the optimal routes).

Your program should be called either Main.py (in Python) or Main.java (in Java), so that it is called with either:

```
  python3 Main.py <MapFilename>
```

or

```
  java Main <MapFilename>
```

The output of your program should contain the following information from running each search algorithm on the inputted map:

1. The length of the path found (i.e., the number of actions your car needs to take to get to the first goal found)
2. The cost of the path
3. Which goal was reached, as a (latitude, longitude) point
4. The number of nodes expanded during search
5. The time spent searching

An example output of your program might be:

```
Map: map0.dat

BFS Path Length: 63
BFS Path Cost: 0.08280000000000001
BFS Goal Reached: (41.2904548, -82.2184917)
Nodes Expanded with BFS: 1274
Time Spent with BFS: 0.019654035568237305

UCS Path Length: 64
UCS Path Cost: 0.08240000000000001
UCS Goal Reached: (41.2904548, -82.2184917)
Nodes Expanded with UCS: 982
Time Spent with UCS: 0.02384805679321289

A* Path Length: 64
A* Path Cost: 0.08240000000000001
A* Goal Reached: (41.2904548, -82.2184917)
Nodes Expanded with A*: 285
Time Spent with A*: 0.014477014541625977
```

To aid your search, you will be given a model of the search problem represented by the `Problem` class, containing the following important functions:

- `startState()` providing your car's current location as a tuple/`State` `(Latitude, Longitude)`
- `actions(state)` providing a list of neighboring intersections in the map that you can reach in one step from the tuple `state`
- `result(state, action)` providing the location of the intersection in the mapyou will reach after choosing `action` in `state`
- `cost(state, action)` providing the cost of choosing `action` in `state`
- `goal(state)` determining whether `state` is a goal state (i.e., a favorite restaurant)
- `goalStates()` providing a list of all favorite restaurants in the map

Within a README file, you should include:

1. The output of your program on all three networks for each algorithm.
2. A paragraph comparing and contrasting the different algorithms based on your results (including a description of the heuristic you chose to use in A*). Questions to think about include: Did they all find paths to the same goals? Did they find different paths with different costs? How did the amount of work they performed compare?
3. A short paragraph describing your experience during the assignment (what did you enjoy, what was difficult, etc.)
4. An estimation of how much time you spent on the assignment, and
5. An affirmation that you adhered to the honor code

Please remember to commit your solution to your repository on GitHub. You do not need to wait until you are done with the assignment; it is good practice to do so not only after each coding session, but maybe after hitting important milestones or solving bugs during a coding session. ***Make sure to document your code***, explaining how you implemented the three search algorithms.

Bonus (10 points)

In addition to BFS, UCS, and A*, also implement Iterative Deepening Search, then include its output in your README file and compare its performance with the other three algorithms.

Honor Code

Each student is to complete this assignment individually. However, students are encouraged to collaborate with one another to discuss the abstract design and processes of their implementations. For example, please feel free to discuss the pseudocode for each search algorithm to help each other work through issues understanding exactly how the algorithms work. At the same, since this is an individual assignment, no code can be shared between students, nor can students look at each other's code.