(364-1-1441) Foundations of Artificial Intelligence

Problem Set 1: From Sea to Shining Sea

Due: 30/5/2024

You need to submit both code and written answers. Problem 1 is a programmatic one, but there are things there you need to submit in writing. Problem 2 only requires written answers and not programming. You will submit one `q1.py` file containing your code, and one `answers.pdf` file containing your written, typewritten (not a scan of handwriting) answers, in English or Hebrew.

Make sure your code compiles, and the output matches what is requested. Your grader will not debug your code, and if it does not compile or output correctly, they will not be in charge of fixing your errors, even if their cause was a very minor mistake.

Also, to simplify your work process, as well as the grader's, please name your variables and methods in a meaningful way (e.g., name a function `heuristicCalculation` and not `myAwesomeFunction`).

1 Problem 1: Surfin' USA

Note: This same setting will be used in Problem Set 2 as well, so build your code sensibly and in a well-organized manner, as you will need it in the future.

Inspired by the upcoming US elections, we will help campaigners to travel throughout the United States of America. You will receive a list of up to 5 starting places (expressed as US counties of the format "`<name>, <2 letter US state code>`"), as well as a party color (`Red` or `Blue`), and a series of the same number of ending locations and parties. Your goal is to bring each party's campaigners to their destinations by going through adjacent counties (using a CSV file that tells you which are adjacent, see below).

You will get the starting and ending locations each as an array of strings.

For example:

Starting locations: {Blue, Washington County, UT ; Blue, Chicot County, AR ; Red, Fairfield County, CT}

Ending locations: {Blue, San Diego County, CA ; Blue, Bienville Parish, LA ; Red, Rensselaer County, NY}

The example below is how path output should look like. In each line, each county name represents a node in the path. Note that once a destination has been reached, it continues to appear, unchanged, in the output.

```
{Washington County, UT (B) ; Chicot County, AR (B) ; Fairfield County, CT(R)}
{Mohave County, AZ (B) ; Ashley County, AR (B) ; Dutchess County, NY (R)}
```

```
{San Bernardino County, CA (B) ; Union Parish, LA (B) ; Berkshire County, MA (R)}
{Orange County, CA (B) ; Lincoln Parish, LA (B) ; Rensselaer County, NY (R)}
{San Diego County, CA (B) ; Bienville Parish, LA (B) ; Rensselaer County, NY (R)}
```

The different paths here are:

1. Washington County, UT → Mohave County, AZ → San Bernardino County, CA → Orange County, CA → San Diego County, CA
2. Chicot County, AR → Ashley County, AR → Union Parish, LA → Lincoln Parish, LA → Bienville Parish, LA
3. Fairfield County, CT → Dutchess County, NY → Berkshire County, MA → Rensselaer County, NY

Shorter paths are better.

You can bring any of the blue party campaigners to a blue destination and any of red campaigners to a red destination, but you cannot have a blue campaigner end up in a destination of the red party (and vice versa) them up.

In this exercise you will only implement A* algorithm to solve this. However, additional algorithms will be coming, so keep that in mind…

You will write a function in python called `find_path`:

```
def find_path(starting_locations,goal_locations,search_method,detail_output):
```

The function takes 4 variables (in this order of input):

`starting_locations` This is the beginning of your search. You can assume these are all valid counties in the format they appear in the county DB (though you can write a function to check this, which will probably help in your debugging).

`goal_locations` These are the locations (and parties) your campaigners need to reach from the starting locations. You can assume these are all valid locations (again, a function checking its legality will probably be helpful to you)

`search_method` This will be an integer. It will have multiple possible values added in Problem Set 2, but for now you will only implement:

1. An A*-heuristic search. **You choose the heuristic**. In your submitted answers, containing the answers to the questions, you will detail your heuristic. It cannot be trivial (e.g., all 0). You need to explain in your submitted answers if your heuristic is admissible, consistent, or neither. A particularly creative or good heuristic may get a small bonus, as well as the fastest running heuristic (i.e., out of all the fully correct ones, the fastest running one will get a small bonus).

`detail_output` This is a binary variable. When it is false, your output is like the text above – you give the full chain of locations. The first one contains the *starting locations*, with party affiliation of the campaigner (`B` or `R`) in parenthesis, and every following line a list of counties that can be legally reached by a single step from the location line before it, until the last line contains the *goal locations*. If no path was found, the output is `No path found`.

If the binary variable is true, for the **first transformation** (from the first set of locations to your second set of locations) you need to print out your work process, so for search method:

A*-heuristic search Print out the heuristic value of the locations you are choosing. So the beginning of your print out will be (remember, this is only done once, for the first choice):

```
{Washington County, UT (B) ; Chicot County, AR (B) ; Fairfield County, CT(R)}
{Mohave County, AZ (B) ; Ashley County, AR (B) ; Dutchess County, NY (R)}
Heuristic: {<your number here> ; <your number here> ; <your number here>}
```

(the rest of the output as above)

Again, for the additional algorithms to come, additional details will be needed here, and they will appear in Problem Set 2.

In order to know which counties are adjacent, you can use the file `adjacency.csv` downloadable from the course website. Each line contains a pair of adjacent counties (be careful, the file includes each county as adjacent to itself!).

2 Problem 2: Search

Describe a state space in which iterative deepening search performs much worst than DFS (for example, time complexity of $O(n^2)$ vs. $O(n)$).