(364-1-1441) Foundations of Artificial Intelligence

Problem Set 2: From Sea^2 to Shining Sea^2

Due: 20/6/2024

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. Problems 2 and 3 only require 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, Part II

Note: Problem description identical to previous problem set

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. **This is just from the previous exercise, it won't be checked for correctness.** An A*-heuristic search. You choose the heuristic. It cannot be trivial (i.e., all 0).
2. A hill climbing algorithm. It should be restarted 5 times (if it didn't find the answer), using adjacent states to the original starting point.

3. Simulated annealing. **You choose the temperature**. In your submitted answers, containing the answers to the questions, you will detail your temperature schedule. It should not go longer than t=100.
4. A local beam search, with the number of beams (k) being 3.
5. A genetic algorithm. Population size is 10. In your submitted answers, containing the answers to the questions, explain the idea of your genetic algorithm (how are you representing solutions, how do you assess solution quality, how do you combine solutions and what is your mutation).

`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)

Hill climbing No change in output.

Simulated annealing Instead of the heuristic, show every action you consider and if the probability of taking it.

Local beam Instead of the heuristic, show your "bag" of actions considered at each stage.

Genetic algorithm Instead of the heuristic, print out your new population.

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

A particularly creative or good heuristic may get a small bonus, particularly with the choice of simulated annealing temperature and genetic algorithm implementation.

2 Problem 2: CSP I

The 4-queen problem involves 4 queens on a 4x4 board, which should not threaten each other under chess rules. Formulate the problem as a CSP, and draw its constraints graph.

3 Problem 3: CSP II

What is the time complexity of the AC-3 algorithm we saw in class? Explain it fully, including your choice of the parameters you use to express the complexity.
(The course book has the answer for this, but you need to explain it in your own words)