Assignment 3: Heuristic Search

CSE 415: Introduction to Artificial Intelligence
The University of Washington, Seattle, Autumn 2019

The reading for this assignment is *Search* in *The Elements of Artificial Intelligence with Python*.

Due Friday, October 18 at 11:59 PM. This is an **individual-work** assignment and students must work independently on the actual coding and reports.

Introduction

This assignment is about the A* algorithm and its use. It's partly about implementing an A* search algorithm, and then mainly about the design and testing of heuristics that can make a search more intelligent. You'll apply it to a route-finding problem and then to various instances of the Eight Puzzle.

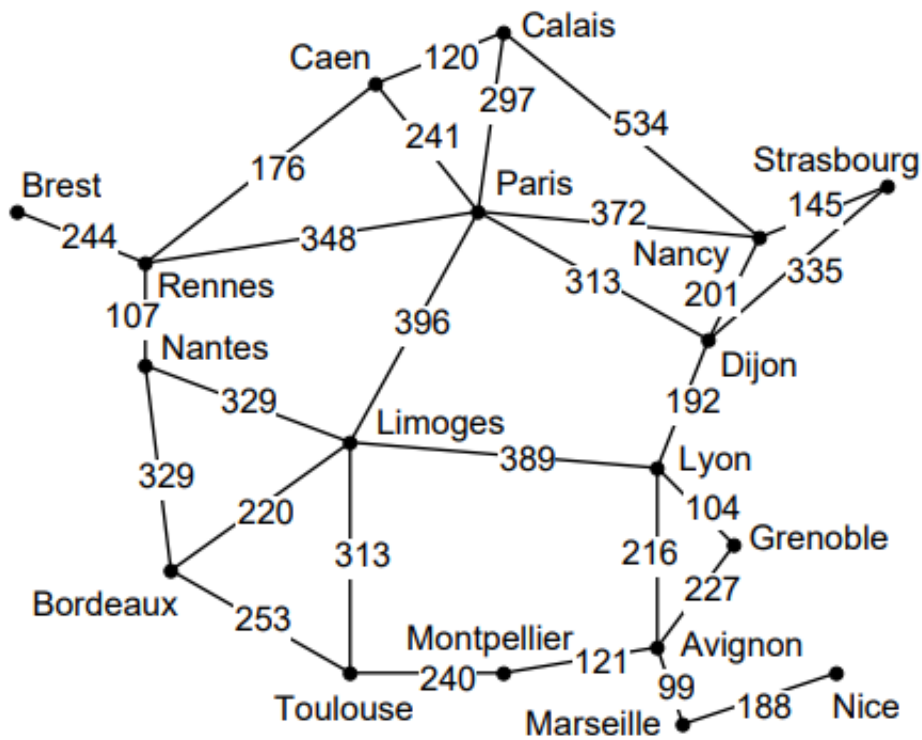
Working with the Starter Code

In Assignment 2 you completed coding up a file to implement the graph-search algorithm known as Uniform-Cost Search. Assuming you followed the guidelines for that assignment, you can run your UCS.py program on the French cities route-finding problem by typing the following on a command line in Darwin, Linux, Cygwin:

```
python3 UCS.py FranceWithCosts
```

If you just put all the files in the same folder and use IDLE or PyCharm, that should work. If you have any trouble with this please see a TA or post in Piazza.

Page 184 in the reading shows the state space for this problem, with the distances on the graph edges. When you run the code you should get a solution path of 4 edges (5 cities) and total distance 1041: [’Rennes’, ’Nantes’, ’Limoges’, ’Lyon’, ’Avignon’]. It might be formatted differently in the printout. The map is also shown below:



Map showing French cities (which serve as the states of this problem) and distance values on roads (represented as graph edges) between them.

The starter code is here.

Implementation and Testing

Do the following tasks.

1. A-Star Implementation (30 points).

   Implement an A* search program by modifying a copy of the UCS.py file, renamed AStar.py.

   Develop your A* program to work with the heuristic function defined in the file FranceWithDXHeuristic.py. Your program will be operated as follows:

   ```
   python3 AStar.py FranceWithDXHeuristic
   ```

   When it is working, it should find the same path that UCS does, but it should have only expanded 12 states instead of 15.

   Here are some hints about how to modify your UCS.py to implement A*. Change the names of the file and the algorithm in the code and the comments. After the code that imports the Problem, globally assign h = Problem.h, so that you can call the heuristic function easily in your algorithm. Within the search algorithm, any time UCS uses gs or new_g as a priority value, you should compute f (or call it fs or new_f, depending on the context), and use the f value as the priority.

When checking the CLOSED list, don't necessarily delete the new state if the same city is already on CLOSED, but look at the priority, and if the new state has a lowever priority value, delete the state on CLOSED and put the new one on OPEN.

2. (10 points) The Eight Puzzle is a simplified version of the popular Fifteen Puzzle. A photo of one commercial model of the Fifteen Puzzle is given below on the left, and a picture of the Eight Puzzle is on the right. (For image credits, see the acknowledgments at the bottom of this page.)

Try to solve some instances of the Eight Puzzle using your UCS.py from Assignment 2. There is a default instance you should try first. You can do it with the command line:

```
python3 UCS.py EightPuzzle
```

There is an easy way to try more instances. Simply provide a string on the command line that represents the initial state in list form.

```
python3 UCS.py EightPuzzle '[[3, 1, 2], [4, 5, 0], [6, 7, 8]]'
```

Another way is to create a file that first imports EightPuzzle's definitions and then overwrites the definition of CREATE_INITIAL_STATE. This is illustrated in the file puzzle_rotate_2.py. To run your UCS.py on such a puzzle, type this:

```
python3 UCS.py puzzle_rotate_2
```

This instance of the Eight Puzzle is named this way, because the solution requires moving all the outside pieces "around" the middle square a distance of two steps each. The lowest-cost path for this instance should have total cost 14.0, since there are seven tiles that move, and they each move twice. Note, if you prefer to do your testing by creating extra files like puzzle_rotate_2, DO

NOT turn those files in with your solutions. Just report the results you get as part of your txt or PDF report file.

Example puzzles to try:

```
# Puzzle A. (should be very fast)
python3 UCS.py EightPuzzle '[[3,0,1],[6,4,2],[7,8,5]]'

# Puzzle B. (should not take long)
python3 UCS.py EightPuzzle '[[3,1,2],[6,8,7],[5,4,0]]'

# Puzzle C. (May take a few minutes)
python3 UCS.py EightPuzzle '[[4,5,0],[1,2,8],[3,7,6]]'

# Puzzle D. (May take several minutes)
python3 UCS.py EightPuzzle '[[0,8,2],[1,7,4],[3,6,5]]'
```

3. (30 points) Implement the following two pre-defined (i.e., fairly standard) heuristics for the Eight Puzzle. (a) Hamming Distance (count the number of tiles out of place, but not the blank, in order to maintain admissibility), and (b) Total of Manhattan distances for the 8 tiles. Each of these should be in a separate file. If you look at the starter code file FranceWithDXHeuristic.py, it shows you how your file should import the basic problem formulation module (EightPuzzle) and then define the heuristic function h. When you are ready to test your Hamming distance heuristic, for example, you will type something like this to test it out.

```
# Test with Puzzle A:
python3 AStar.py EightPuzzleWithHamming '[[3,0,1],[6,4,2],[7,8,5]]'
```

If your A* algorithm and heuristics are working correctly, then you should see some rather noticeable improvements in the search speed and statistics.

4. Comparing Heuristics and Writing the Report (30 points). Test these heuristics on the four previously given instances of the Eight Puzzle, and record the results. Also, compare them with "no heuristic" (straight Uniform Cost Search).

   You'll be turning in a "report" for this assignment in addition to your code (see below). Create a report section called "Heuristics for the Eight Puzzle". Compare the performance of these heuristics on each of the example puzzles (given above). For each puzzle-heuristic pair, report (a) whether the puzzle was successfully solved, (b) length of the solution path found, in number of edges, (c) total cost of the path found, (d) number of states expanded, and (d) MAX_OPEN_LENGTH. Put this information into the table, described below.

   The table should have a row for each puzzle instance and heuristic (e.g., "(Puzzle A, Hamming)", and columns for the following: puzzle instance permutation (e.g., [3,0,1,6,4,2,7,8,5]; this flattened list is OK here), success (yes/no), count of expanded nodes, aborted (yes/no). If it takes more than 5 minutes to solve a particular problem with a particular heuristic then note that; you may abort such runs. You are welcome to use the format for your report: shown in the starter-code file named `a3-report-template.txt`.

Optional Extra Credit (up to 15 points)

This is probably a lot more work per point than in the rest of the assignment, but it's a fun exercise if you have time.

(a -- 5 points) Create a formulation of the 2x2x2 Rubik cube that works with UCS.py. You can limit the operators to these six: F (front), B (back), U (upper), D (down-side), L (left), R (right). Each operator rotates the indicated face by 90 degrees clockwise. That's clockwise looking at the cube from the outside, as if that face is facing you.

(b -- 5 points) Implement two heuristic evaluation functions for the 2x2x2 Rubik cube.

One should be a Hamming distance similar to that used in the Eight Puzzle; given a state, it will return the number of faces that are not where they need to be in the goal. The other is one that you should design yourself. It should be more informed, but you should design it to be admissible. One way to approach this is to consider what a Manhattan-like distance would be for cubie faces in this puzzle.

(c -- 5 points) Evaluating Your Custom Heuristics ...

Create a report item: "Evaluating my Custom Heuristic." (a) First explain how your heuristic works, and the thinking behind it. (b) Second tell whether it actually outperforms any of the other heuristics in terms of lowering the number of expanded states while still solving the problem. (c) Finally, discuss how you believe the computational cost of computing your heuristic compares with the cost of computing the others.

Notes about Grading

The staff is planning to autograde parts of your code. To do this, it is likely that your files will be imported as modules by the autograder. Your search should not automatically start running when the module is imported. See the starter code UCS.py for how it uses the test "if __name__ == '__main__' " near the bottoom of the file to avoid running the search automatically if it is being imported, but still run automatically if it is the main program. You'll be fine if you simply carry over this feature from UCS.py when you create your copy of it to turn it into your implementation of A*. The autograder will be comparing the solutions and statistics found by your code with expected solutions, by calling functions in your code and examining the values of global variables in your code. These globals are generally already in place for you in the starter code UCS.py. So you should not rename variables like SOLUTION_PATH or any of the other global variables..

Commenting your Code

Each of your Python files should begin with a multiline string that gives, on the first line, your name and UWNetID. It should identify the file (name and purpose), and explain if it is a modified version of starter code in CSE 415 or is a new file you created from scratch.

Follow reasonable commenting practice in your code. The comments in the starter code can serve as examples of the commenting style that we consider appropriate.

Files to Turn In

Here is a list of the files to include in your Canvas turn-in. Do NOT zip up the files, since that will interfere with the grading workflow. (To incentivize compliance on this, the staff will be deducting 2 points, if the files are zipped.)

```
AStar.py
EightPuzzleWithHamming.py
EightPuzzleWithManhattan.py
A3_Report.pdf
optional: Rubik2Cube.py
optional: Rubik2WithHeuristics.py
```

Updates and Corrections

If necessary, updates and corrections will be posted here and/or mentioned in class or in ED.

Acknowledgments:

The Fifteen Puzzle image is courtesy of Wayne Schmidt (http://waynesthisandthat.com/15puzzle.htm). The Eight Puzzle image comes from https://cs.stackexchange.com/questions/16515/reachable-state-space-of-an-8-puzzle.