CMSC 671 (Introduction to AI) – Fall 2017

Homework 1: Python and AI

Due: 9/18 at 11:59pm.

Turnin: Blackboard.

Submit:
- Parts I and II together as a **single PDF file** named *yourlastname*_hw1-text.pdf.
- Part III as a **single .py file** named *yourlastname*_hw1-program.py, containing everything specified in the assignment.

Notes:
- These are individual assignments, not group work.
- All files must start with your last name.

PART I. WHAT IS AI? (15 PTS)

Reading: Read John McCarthy's paper, "What is AI?"
http://tiny.cc/mc-what-is-ai

Reading: Read the recent 100-year retrospective *through section II*:
http://ai100.stanford.edu/2016-report

Assignment: Answer the following questions in a short *essay* (500-800 words):

- Based on *both* papers:
  - What did you think "AI" meant before the reading?
  - Did anything you read change your mind? How?
- On McCarthy:
  - Does McCarthy see the primary goal of AI as modeling human intelligence? Why or why not?
- On the Stanford report:
  - How much of "modern AI" is based on trying to model human intelligence?
  - Do you agree on what the primary goal(s) of AI should be, and whether it is achievable?
- Summarize some of the key challenges in achieving human-level intelligence.

PART II. AI NOW (15 PTS.)

Assignment: Answer the following questions (1-2 sentences per question)

1. What current research trend do you think shows the most *promise*, that is, seems most likely to produce interesting and important results?
2. Why is that your choice?
3. What current area (research trend OR application area) of AI are you most excited about?
4. Why?
5. Based on what you read, what area or trend do you think is *least* promising, or should not be pursued?
6. Why?

PART III. INTRODUCTION TO PYTHON (40 PTS)

If you are not familiar with Python, there are lots of online resources. Any resource where you look at code samples should be cited in the comments at the beginning of the program.

Documentation and error checking are essential in this class, so although these problems are very simple, your code must be documented, and error cases must be handled. (For example, what if someone passes a non-coordinate string to 4(b)?)

We will test your code automatically, so it is important that you follow all naming conventions specified and use the correct parameters.

We are using **Python 3, not 2**, for this course.

Problem 1: Lists, Sets, Tuples, and Libraries (12 points)

(a) Import the `random` library.

(b) Write a short function called `world_shuffle` that: *3*

1. Populates a list with the individual characters of "Hello World!" ('H', 'e', 'l', 'l', etc.)
2. Uses `random.shuffle` to randomly permute the elements of the list.
3. Concatenates all the elements of the shuffled list into a single string; and
4. Returns the result.

Here is an example of one possible output:

```
world_shuffle() ⇒ rleWHoold !l
```

(c) Write a short function called `shuffle_anyset` that: *4*

1. Creates an empty set.
2. Populates the set with the individual characters of "Hello World!" ('H', 'e', 'l', 'l', etc.)
3. Concatenates the elements of the set into a string; and
4. Returns the result.

(d) Write a short function called `make_map` that: *5*

1. Creates an empty list.
2. Populates the list with 25 two-element tuples representing *pairs* of numbers ranging from 1-5 and 1-5. (I.e.: `(1,1)`, `(1,2)`, … `(1,5)`, `(2,1)`, `(2,2)`, …, `(5,5)`.)
3. Prints the resulting list, five items per line.

Here are the first two lines of the expected output:

```
(1, 1) (1, 2) (1, 3) (1, 4) (1, 5)
(2, 1) (2, 2) (2, 3) (2, 4) (2, 5)
```

These tuples are coordinates into a grid-based map (shown below).

Problem 2: Dictionaries and Manhattan Distance (18 points)

(a) Write a function called `map_dict` that: *6*

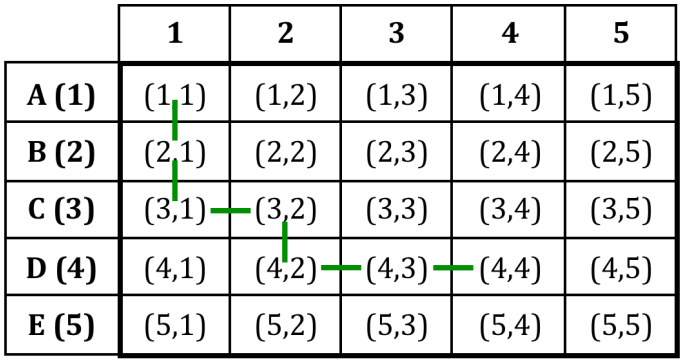
1. Creates a dictionary containing 25 key/value pairs labeling the map spaces.
    a. Key: string label, e.g., "A1", "A2", …, "E5"
    b. Value: the tuple containing the tuple of coordinates for that space.
    c. Returns this dictionary.

(b) Write a function called `manhattan_dist` that: *12*

1. Takes two map coordinate labels as arguments.

2. Calculates the *Manhattan distance* between those coordinates: the smallest number of steps you would have to take to get between them if you cannot go diagonally.
3. Returns this distance.

Here is an example of one possible test case:
manhattan_dist("A1", "D4") ⇒ 6



Problem 3: Call and Test (10 points)

Write a main method that calls all of the functions from Problems 1 and 2, and prints out the results with descriptive labels.