**CS 480 Python Assignment 1**

**Assigned: 6 February 2023   Due: 22 February 2023**

This is a Python programming assignment. You can either do this assignment individually or you can do it in a pair with one other student from the class. If you choose to do it in a pair, please upload one (1) copy of (each part of) the program with both of your names/ids clearly indicated in a comment in the file (but only one ID in the upload page).

This assignment involves starting with existing Python code and extending it. When you are done, please [**upload**](http://vh216602.truman.edu/agarvey/cs480/uploadsp23.php) your files. You should not change any of the files given here. You should just create new files, one for the RiverCross problem and one for the Eight Puzzle problem. Your files should **import** the appropriate files to allow you to write your code.

1. A general state space search algorithm is implemented in [**search.py**](http://vh216602.truman.edu/agarvey/cs480/sp07/search.py). Go through this and try to understand it. An example of how to implement a particular search problem using the state space code is shown in [**pitcher.py**](http://vh216602.truman.edu/agarvey/cs480/sp07/pitcher.py). Note in particular how the *PitcherState* class works in conjunction with the *Search* class. Using the *PitcherState* class as a model, write a *Mover* class that implements a solution to the following important problem. **Note that this problem has changed as of Feb. 10, 2023.**

*A mover has to move an elephant, a really big mouse and a wheel of cheese from Point A to Point B. Only one item fits in the moving van at a time along with the mover. The mover can't leave the elephant alone with the mouse (or it will die of fright) and can't leave the mouse alone with the cheese (because it will eat it). Formulate the actions for this problem and implement them.*

1. Save the file [**pq.py**](http://vh216602.truman.edu/agarvey/cs480/sp07/pq.py) to your own directory. This contains the definition of a PriorityQueue class. Read through the implementation and see if you understand how it works. You can test it by modifying the test code at the bottom of the file, but you don't really need to completely understand this file.

Save the file [**informedSearch.py**](http://vh216602.truman.edu/agarvey/cs480/sp07/informedSearch.py)

to your own directory. This contains updated definitions of the classes we used in the previous question. The InformedNode class now takes a goal state and has an added method called priority. The InformedSearch class now uses a priority queue and creates instances of the InformedNode class in its execute method. It also keeps track of how many nodes it has expanded during the search process and reports this with the solution. The InformedProblemState now has an added method called heuristic which takes a goal state and returns an estimate of the distance from the current state to the goal state.

**Create a file that implements the states, operators, and heuristics needed for the eight puzzle problem.** Your EightPuzzle class should inherit from the InformedProblemState class. *Remember to make sure that your operators make a copy of the current state before moving the tiles.*

Test your solution on the following starting states A-H using the same goal each time. State A should take 2 steps, state B should take 6 steps, and state C should take 8 steps. You'll need to determine the length of the other solutions.

1 2 3 1 3 1 3 4 1 3 7 1 2 8 1 2 2 6 3 7 3 4 7 4 5

8 4 8 2 4 8 6 2 4 2 5 8 3 7 4 4 5 6 1 5 6 3

7 6 5 7 6 5 7 5 8 7 6 6 5 4 6 5 3 1 8 7 8 2 8 1 2

goal A B C D E F G H

In order to demonstrate that A\* is superior to standard BFS and that the Manhattan distance heuristic is more informed than the tiles out of place heuristic, you will compare the number of nodes that each search expands on the problems given above (A-H). The only change you'll need to make to do a BFS search rather than an A\* search is to replace the priority queue with the standard queue that we used in the previous question. One easy way to do this is to always have a heuristic, but in BFS have the heuristic always return 0. Convince yourself that this is equivalent to BFS. Inside a comment in your eight puzzle file, create and fill in the following table:

Node Expansions

Problem BFS A\*(tiles) A\*(dist)

A

B

C

D

E

F

G

H