B551 (online) - Assignment 1: Searching

Fall 2016
Due: Sunday October 2nd, 11:59PM
(You may submit up to 48 hours late for a 10% penalty.)

This assignment will give you practice with posing AI problems as search, and with un-informed and informed search algorithms. This is also an opportunity to dust off your coding skills.

You'll work in a group of 2 people for this assignment; we've already assigned you to a group (see details below). You should only submit **one** copy of the assignment for your team, through GitHub, as described below. All people on the team will receive the same grade on the assignment, except in unusual circumstances; we will collect feedback about how well your team functioned in order to detect these circumstances. Please read the instructions below carefully; we cannot accept any submissions that do not follow the instructions given here. Most importantly: please **start early,** and ask questions on Piazza or in office hours.

Academic integrity. You and your teammates may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about Python syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials (e.g. in source code comments). However, the work and code that you and your partners submit must be your group's own work, which the two of you personally designed and wrote. You may not share written answers or code with any other students except your own partner, nor may you possess code written by another student who is not your partner, either in whole or in part, regardless of format.

What to do. The following problems require you to write programs in Python. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and basic data structures like queues. You must write all of the rest of the code yourself. If you have any questions about this policy, please ask us. We recommend using the CS Linux machines (e.g. `burrow.soic.indiana.edu`). You may use another development platform (e.g. Windows), but make sure that your code works on the CS Linux machines before submission.

For each problem, please write a detailed comments section at the top of your code that includes: (1) a description of how you formulated the search problem, including precisely defining the state space, the successor function, the edge weights, and (if applicable) the heuristic function(s) you designed, including an argument for why they are admissible; (2) a brief description of how your search algorithm works; (3) and discussion of any problems you faced, any assumptions, simplifications, and/or design decisions you made.

You'll submit your code via GitHub. We strongly recommend using GitHub as you work on the assignment, pushing your code to the cloud whenever you reach a milestone. If you have not used IU GitHub before, instructions for getting started with git are available on the Canvas and there are many online tutorials.

0. For this project, we are assigning you to a team. We will let you change these teams in future assignments. You can find your assigned teammate(s) by logging into IU Github, at `http://github.iu.edu/`. In the upper left hand corner of the screen, you should see a pull-down menu. Select `cs-b551-fa2016`. Then in the yellow box to the right, you should see a repository called *userid1-userid2*-a1 or *userid1-userid2-userid3*-a1, where the other user ID(s) correspond to your teammate(s).

   To get started, clone the github repository:

   `git clone https://github.iu.edu/cs-b551-fa2016/`*your-repo-name*`-a1`

   where *your-repo-name* is the one you found on the GitHub website above. (If this command doesn't work, you probably need to set up IU GitHub ssh keys. See Canvas for help.)

1. It's September, which means you have only 6 months to make your Spring Break vacation plans to a dream destination! We've prepared a dataset of major highway segments of the United States (and parts of southern Canada and northern Mexico), including highway names, distances, and speed

limits; you can visualize this as a graph with nodes as towns and highway segments as edges. We've also prepared a dataset of cities and towns with corresponding latitude-longitude positions. These files should be in your GitHub repo from when you cloned in step 0. Your job is to implement algorithms that find good driving directions between pairs of cities given by the user. Your program should be run on the commandline like this:

```
python route.py [start-city] [end-city] [routing-option] [routing-algorithm]
```

where:

- `start-city` and `end-city` are the cities we need a route between.
- `routing-option` is one of:
  - `segments` finds a route with the fewest number of "turns" (i.e. edges of the graph)
  - `distance` finds a route with the shortest total distance
  - `time` finds the fastest route, for a car that always travels at the speed limit
  - `scenic` finds the route having the least possible distance spent on highways (which we define as roads with speed limits 55 mph or greater)
- `routing-algorithm` is one of:
  - `bfs` uses breadth-first search
  - `dfs` uses depth-first search
  - `ids` uses iterative deepening search
  - `astar` uses A* search, with a suitable heuristic function

The output of your program should be a nicely-formatted, human-readable list of directions, including travel times, distances, intermediate cities, and highway names, similar to what Google Maps or another site might produce. In addition, the *last* line of output should have the following machine-readable output about the route your code found:

```
[total-distance-in-miles] [total-time-in-hours] [start-city] [city-1] [city-2] ... [end-city]
```

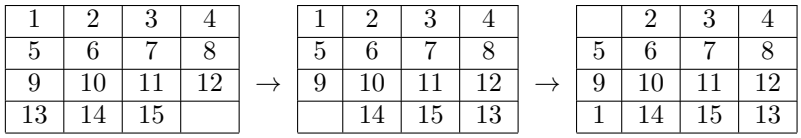
Please be careful to follow these interface requirements so that we can test your code properly. For instance, the last line of output might be:

```
51 1.0795 Bloomington,_Indiana Martinsville,_Indiana Jct_I-465_&_IN_37_S,_Indiana Indianapolis,_Indiana
```

Like any real-world dataset, our road network has mistakes and inconsistencies; in the example above, for example, the third city visited is a highway intersection instead of the name of a town. Some of these "towns" will not have latitude-longitude coordinates in the cities dataset; you should design your code to still work well in the face of these problems.

In the comment section at the top of your code file, please include a brief analysis of the results of your program, answering the questions: (1) Which search algorithm seems to work best for each routing options? (2) Which algorithm is fastest in terms of the amount of computation time required by your program, and by how much, according to your experiments? (To measure time accurately, you may want to temporarily include a loop in your program that runs the routing a few hundred or thousand times.) (3) Which algorithm requires the least memory, and by how much, according to your experiments? (4) Which heuristic function did you use, how good is it, and how might you make it better? (5) Supposing you start in Bloomington, which city should you travel to if you want to take the longest possible drive (in miles) that is still the shortest path to that city? (In other words, which city is furthest from Bloomington?)

2. Consider a variant of the 15-puzzle, but with the following important change: when the empty tile is on the edge of the board, the tile on the opposite side of the board can be slid into the opening. For example, the following are valid moves:



The goal of the puzzle is to find the shortest possible sequence of moves that restores the canonical configuration (on the left above) given an initial board configuration. Write a program called solver15.py that finds a solution to this problem efficiently using A* search. Your program should run on the command line:

```
python solver15.py [input-board-filename]
```

where `input-board-filename` is a text file containing a board configuration in a format like:

```
5 7 8 1
10 2 4 3
6 9 11 12
15 13 14 0
```

where 0 is the position of the empty tile.

The program can output whatever you'd like, except that the last line of output should be a machine-readable representation of the solution path you found, in this format:

```
[move-1] [move-2] ... [move-n]
```

where each move is encoded as a letter `L`, `R`, `U`, or `D` for left, right, up, or down, respectively, indicating which direction a tile is slid in order to fill the missing tile. For instance, the two moves in the picture above would be represented as:

```
L U
```

Try to make your solver as fast as possible. Which heuristic functions did you try, and which works best?

What to turn in

Turn in the two programs on GitHub (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59PM on the due date. To make sure that the latest version of your work has been accepted by GitHub, you can log into the github.iu.edu website and browse the code online.