CS B551 - Assignment 1: Searching

Fall 2022
Due: Monday, October 10, 11:59:59PM Eastern (New York) time
Late submissions accepted until Sunday October 9, 11:59:59PM, with 10% grade 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. 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 Q&A Community or in office hours.

Guidelines for this assignment

Coding requirements. For fairness and efficiency, we use an automatic program to grade your submissions. This means you must write your code carefully so that our program can run your code and understand its output properly. In particular:

1. You must code this assignment in Python 3, not Python 2.
2. Make sure to use the program file name we specify.
3. Use the skeleton code we provide, and follow the instructions in the skeleton code (e.g., to not change the parameters of some functions).
4. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and data structures like queues, as long as they are already installed on silo.sice.indiana.edu.
5. **IMPORTANT:** In addition to testing your programs on your own, test your code on silo.sice.indiana.edu using test scripts that are provided in the Github repo. To run these test scripts on your code, type:
   `python3 -m pytest -v`
   This will automatically run your programs on some sample test cases, and indicate whether your programs passed or failed. These test cases are just samples; we will run this on more cases while grading, so make sure to test your code thoroughly.

For each of these problems, you will face some design decisions along the way. Your primary goal is to write clear code that finds the correct solution in a reasonable amount of time. To do this, you should give careful thought to the search abstractions, data structures, algorithms, heuristic functions, etc.

Groups. You'll work in a group of 1-3 people for this assignment; we've already assigned you to a group as indicated in a text file attached to this assignment. You should only submit **one** copy of the assignment for your team, through GitHub, as described below. All the people on the team will receive the same grade, except in unusual circumstances; we will collect feedback about how well your team functioned in order to detect these cir-cumstances. If you do not wish to work in a group, please contact instructors through Canvas and request that you work alone.

Coding style and documentation. We will not explicitly grade based on coding style, but it's important that you write your code in a way that we can easily understand it. Please use descriptive variable and function names, and use comments when needed to help us understand code that is not obvious.

Report. *Please put a report describing your assignment in the Readme.md file in your Github repository.* For each problem, please include: (1) a description of how you formulated the search problem, including precisely defining the state space, the successor function, the edge weights, the goal state, 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. These comments are especially important if your code does not work as well as you would like, since it is a chance to document the energy and thought you put into your solution. For example, if you tried several different heuristic functions before finding one that worked, feel free to describe this in the report so that we appreciate the work that you did.

Academic integrity. We take academic integrity very seriously. To maintain fairness to all students in the class and integrity of our grading system, we will prosecute any academic integrity violations that we discover. *Before beginning this assignment, make sure you are familiar with the Academic Integrity policy of the course, as stated in the Syllabus, and ask us about any doubts or questions you may have.*

Part 0: Getting started

For this project, you can find your team arrangement by logging into IU Github at `https://github.iu.edu/cs-b551-fa2022/` and navigating to the repository `a1-release`. You should submit your assignment in a repository that follows this naming convention: *userid1* -a1, *userid1-userid2* -a1, or *userid1-userid2-userid3* -a1, where the other user ID(s) correspond to your teammate(s). Now that you know your teammate(s) userid(s) and last names, you can write them an email at userid@iu.edu.

To get started, clone the github repository:

```
git clone git@github.iu.edu:cs-b551-fa2022/a1-release-res
```

If that doesn't work, instead try:

```
git clone https://github.iu.edu/cs-b551-fa2022/a1-release-res
```

(If neither command works, you probably need to set up IU GitHub ssh keys. See Canvas for help.)

Part 1: Birds, heuristics, and A*

On a power line sit five birds, each wearing a different number from 1 to N. They start in a random order and their goal is to re-arrange themselves to be in order from 1 to N (e.g., 12345), in as few steps as possible. In any one step, exactly one bird can exchange places with exactly one of its neighboring birds. We can pose this as a search problem in which there is a set of states S corresponding to all possible permutations of the birds (i.e., S = {12345, 12354, 12453, ...} for N = 5).

We provided a skeleton code to this problem implemented using BFS: `solve_birds.py`. You can run it with the command line:

```
python3 solve_birds.py birds.txt
```

But this code is very slow. Your job is to convert it into a fast A* solution by modifying the functions `h()` and `solve()`. To do this:

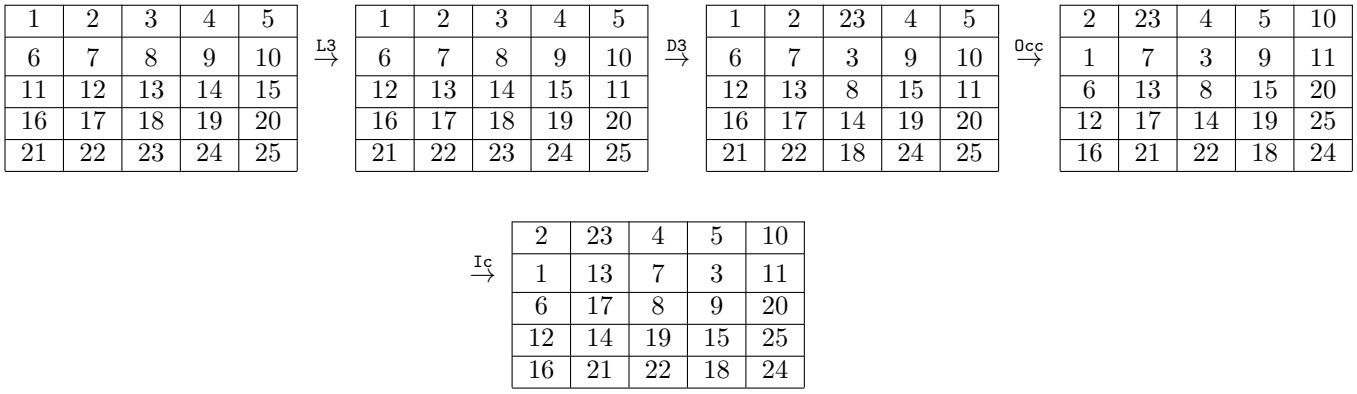
1. Switch from a queue to a priority queue.
2. Implement an admissible heuristic h(s).
3. For the priority value, use f(s) = g(s) + h(s), where g(s) is the cost from the initial state to s.

This is a warm-up problem, and your goal is to just solve all cases in `birds.txt` under 10 seconds total. We will not run additional tests on this one.

Part 2: The 2022 Puzzle

Consider the 2022 puzzle, which is a lot like the 9-puzzle we talked about in class, but: (1) it has 25 tiles, so there are no empty spots on the board; (2) instead of moving a single tile into an open space, a move in this puzzle consists of either (a) sliding an entire row of tiles left or right one space, with the left- or right-most tile 'wrapping around' to the other side of the board, (b) sliding an entire column of tiles up or down one space, with the top- or bottom-most tile 'wrapping around' to the other side of the board, (c) rotating the outer 'ring' of tiles either clockwise or counterclockwise, or (d) rotating the inner ring either clockwise or counterclockwise.

For example, here is a sequence of three moves on such a puzzle:



The goal of the puzzle is to find a short sequence of moves that restores the canonical configuration (on the left above) given an initial board configuration. We've provided skeleton code to get you started. You can run the skeleton code on the command line:

```
python3 solver2022.py [input-board-filename]
```

where `input-board-filename` is a text file containing a board configuration (we have provided an example). You'll need to complete the function called `solve()`, which should return a list of valid moves. The moves should be encoded as strings in the following way:

- For sliding rows, `R` (right) or `L` (left), followed by the row number indicating the row to move left or right. The row numbers range from 1-5.
- For sliding columns, `U` (up) or `D` (down), followed by the column number indicating the column to move up or down. The column numbers range from 1-5.
- For rotations, `I` (inner) or `O` (outer), followed by whether the rotation is clockwise (`c`) or counterclockwise (`cc`).

For example, the above diagram performs the moves `L3` (slide row 3 left), `D3` (slide column 3 down), `Occ` (outer counterclockwise), and `Ic` (inner clockwise).

The initial code does not work correctly. Using this code as a starting point, implement a fast version, using A* search with a suitable heuristic function that guarantees finding a solution in as few moves as possible. Try to make your code as fast as possible even for difficult boards, although it is not necessarily possible to quickly solve all puzzles. For example, `board1.txt` can be solved in 11 moves. You will need to be creative with your heuristic function in order to find this solution in less than 15 minutes.

In your report, answer the following questions:

1. *In this problem, what is the branching factor of the search tree?*
2. *If the solution can be reached in 7 moves, about how many states would we need to explore before we found it if we used BFS instead of A* search? A rough answer is fine.*

In addition to doing your own testing, it is important that you test your program on silo.sice.indiana.edu using our test script to ensure that we will be able to run it and grade it accurately.

Part 3: Road trip!

It's a great time to start planning a post-pandemic road trip! If you stop and think about it, finding the shortest driving route between two distant places — say, one on the east coast and one on the west coast of the U.S. — is extremely complicated. There are over 4 million miles of roads in the U.S. alone, and trying all possible paths between two places would be nearly impossible. So how can mapping software like Google Maps find routes nearly instantly? The answer is A* search!

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. Your job is to find good driving directions between pairs of cities given by the user.

The skeleton code can be run on the command line like this:

```
python3 ./route.py [start-city] [end-city] [cost-function]
```

where:

- `start-city` and `end-city` are the cities we need a route between.
- `cost-function` is one of:
  - `segments` tries to find a route with the fewest number of road segments (i.e. edges of the graph).
  - `distance` tries to find a route with the shortest total distance.
  - `time` finds the fastest route, assuming one drives the speed limit.
  - `delivery` finds the fastest route, in expectation, for a certain delivery driver. Whenever this driver drives on a road with a speed limit ≥ 50 mph, there is a chance that a package will fall out of their truck and be destroyed. They will have to drive to the end of that road, turn around, return to the start city to get a replacement, then drive all the way back to where they were (they won't make the same mistake the second time they drive on that road).

    Consequently, this mistake will add an extra $2 \cdot (t_{\text{road}} + t_{\text{trip}})$ hours to their trip, where t_{trip} is the time it took to get from the start city to the beginning of the road, and t_{road} is the time it takes to drive the length of the road segment.

    For a road of length ℓ miles, the probability p of this mistake happening is equal to $\tanh\left(\frac{\ell}{1000}\right)$ if the speed limit is ≥ 50 mph, and 0 otherwise.[1] This means that, in expectation, it will take $t_{\text{road}} + p \cdot 2(t_{\text{road}} + t_{\text{trip}})$ hours to drive on this road.

For example:

```
python3 ./route.py Bloomington,_Indiana Indianapolis,_Indiana segments
```

You'll need to complete the get_route() function, which returns the best route according to the specified cost function, as well as the number of segments, number of miles, number of hours for a car driver, and expected number of hours for the delivery driver. See skeleton code for details.

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"

[1] This formula is not incredibly special, but what's important is that it increases as the length of the road l increases, and it will always be between 0 and 1, which means it can be interpreted as a probability. You can access the tanh function in Python by using the `math` module: `from math import tanh`

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 addition to doing your own testing, it is important that you test your program on silo.sice.indiana.edu using our test script to ensure that we will be able to run it and grade it accurately.

Extra credit. Implement an additional cost-function: `statetour` should find the shortest route from the start city to the end city, but that passes through at least one city in each of the 48 contiguous U.S. states.

What to turn in

Create a private repository using the naming convention specified above, and keep the same directory structure as in `a1-release` (i.e., separating the programs into Part1, Part2, Part3). Make sure that you stick to this naming scheme and directory structure so that our autograder can locate your submission correctly. And make sure it is private (not internal or public) so others will not see your submission. Turn in the three programs on GitHub (remember to `add`, `commit`, `push`) — we'll grade whatever version you've put there as of 11:59:59PM 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. **Your programs must obey the input and output formats we specify above so that we can run them, and your code must work on the SICE Linux computers.**

Tip: These three problems are very different, but they can all be posed as search problems. This means that if you design your code well, you can reuse or share a lot of it across problems.