CS B551 - Assignment 1: Searching

Fall 2020

Due: Thursday October 8, 11:59:59PM Eastern (New York) time

Resubmissions or late submissions due: Sunday October 11, 11:59:59PM Eastern (New York) time

(See *Submissions* below for details.)

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 Piazza or in office hours.

Guidelines for this assignment

Coding requirements. For fairness and efficiency, we use a semi-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. You should test your code on one of the SICE Linux systems such as burrow.sice.indiana.edu. 3. Your code must obey the input and output specifications given below. 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 the SICE Linux servers. 5. Make sure to use the program file name we specify.

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. To encourage innovation, we will conduct a competition to see which program can solve the hardest problems in the shortest time, and a small amount of your grade (or extra credit) may be based on your program's performance in this competition.

Submissions. As with A0, to help you make sure your code satisfies our requirements, we will give you the chance to revise your code based on some initial feedback from our grading program. If you submit your code by the **Due date** above, we will run your code through a simplified version of our grading program and send you feedback within 24 hours. You may then submit a revised version before the **Resubmission** listed above, with no grade penalty. If you do not submit anything by the **Due date** date, you may still submit by the **Resubmission** deadline, but it will count as a late submission and the 10% grade deduction will apply. You'll submit your code via GitHub.

Groups. You'll work in a group of 1-3 people for this assignment; we've already assigned you to a group (see details below) according to your preferences. 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 circumstances. The requirements for the assignment are the same no matter how many teammates you have, but we expect that teams with more people will submit answers that are significantly more "polished" — e.g., better documented code, faster running times, more thorough answers to questions, etc.

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.* To briefly summarize, you 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). We expect that you'll write your own code and not copy anything from anyone else, including online resources. *However, if you do copy something (e.g., a small bit of code that you think is particularly clever), you have to make it explicitly clear which parts were copied and which parts were your own. You can do this by putting a very detailed comment in your code, marking the line above which the copying began, and the line below which the copying ended, and a reference to the source.* Any code that is not marked in this way must be your own, which you personally designed and wrote. You may not share written answers or code with any other students, nor may you possess code written by another student, either in whole or in part, regardless of format.

Part 0: Getting started

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-fa2020`. Then in the box below, you should see a repository called *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 their userid(s), 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-fa2020/`*`your-repo-name`*`-a1`

If that doesn't work, instead try:

`git clone https://github.iu.edu/cs-b551-fa2020/`*your-repo-name*`-a1`

where *your-repo-name* is the one you found on the GitHub website above. (If neither command works, you probably need to set up IU GitHub ssh keys. See Canvas for help.)

Part 1: The 2020 Puzzle

To commemorate the year 2020 has been — which has been, well, memorable — we're introducing the 20-puzzle. It's a lot like the 15-puzzle we talked about in class, but: (1) it's played on a 5x4 board, (2) it has 20 tiles, so that there are no empty spaces on the board, (3) 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, with the left- or right-most tile "wrapping around" to the other side of the board, or (b) sliding an entire column of the puzzle up or down, with the top- or bottom-most tile "wrapping around." For example, here is a sequence of two moves on such a puzzle:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20

→

1	2	3	4	5
10	6	7	8	9
11	12	13	14	15
16	17	18	19	20

→

1	17	3	4	5
10	2	7	8	9
11	6	13	14	15
16	12	18	19	20

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. Write a program called solver20.py that finds a solution to this problem efficiently using A* search. Your program should run on the command line like:

```
python3 solver20.py [input-board-filename]
```

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

```
1 17 3 4 5
10 2 7 8 9
11 6 13 14 15
16 12 18 19 20
```

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, and a row or column number (indexed beginning at 1). For instance, the moves in the picture above would be:

```
R2 D2
```

We've included some sample code in your repository, but it is quite slow for complicated boards. Using this code as a starting point, implement a faster 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, although it is not necessarily possible to write code that will be able to quickly solve all puzzles.

Part 2: 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. These files should be in the GitHub repo 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 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, for a car that always travels *five miles per hour above* the speed limit.
- `cycling` tries to find the safest route for cycling. (Assume that in each mile of roadway, there is a probability of accident $p(s) = 0.000001s$ where s is the speed limit. The expected number of accidents on a given road segment is then $p(s) \times l$, where l is the length of the road. For example, if the speed limit is 50mpg and a road is 10 miles long, the expected number of accidents on that roadway is 0.0005. The goal is to find a route with the fewest total number of expected accidents).

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 might produce. In addition, the *last* line of output should be the following line summarizing the route you found:

```
[total-segments] [total-miles] [total-hours] [total-expected-accidents] [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:

```
3 51 1.0795 0.002498 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.

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.

Part 3: Choosing teams

In a certain Computer Science course, students are assigned to groups according to preferences that they specify. Each student is sent an electronic survey and asked to give answers to three questions:

1. What is your IU email username?
2. Please choose one of the options below and follow the instructions.
   (a) You would like to work alone. In this case, just enter your userid in the box and nothing else.
   (b) You would like to work in a group of 2 or 3 people and already have teammates in mind. In this case, enter all of your userids (including your own!) in the box below, in a format like userid1-userid2 for a team of 2, or userid1-userid2-userid3 for a team of 3.
   (c) You would like to work in a group of 2 or 3 people but do not have any particular teammates in mind. In this case, please enter your user ID followed by one "zzz" per missing teammate (e.g. djcran-zzz where djcran is your user ID to request a team of 2, or djcran-zzz-zzz for a team of 3).
   (d) You would like to work in a group of 3 people and have some teammates in mind but not all. Enter all of your ids, with zzz's to mark missing teammates (e.g. if I only have one teammate (vkvats) in mind so far, I'd enter djcran-vkvats-zzz).
3. If there are any people you DO NOT want to work with, please enter their userids here (separated by commas, e.g. userid1,userid2,userid3).

Unfortunately, the student preferences may not be compatible with each other: student A may request working with student B, but B may request not to work with A, for example. Students are going to complain, so the course staff decides to try to minimize their own work. The course staff estimates that:

- They need k minutes to grade each assignment, so total grading time is k times number of teams.
- Each student who requested a specific group size and was assigned to a different group size will complain to the instructor, taking 1 minute of the instructor's time.
- Each student who is not assigned to someone they requested sends a complaint email, which will take n minutes for the instructor to read and respond. If a student requested to work with multiple people, then they will send a separate email for each person they were not assigned to.
- Each student who is assigned to someone they requested *not* to work with (in question 3 above) complains to the Dean, who meets with the instructor for m minutes. If a student is assigned to a group with multiple people they did not want to work with, a separate meeting will be needed for each.

The total time spent by the course staff is equal to the sum of these four components. You can assume that each student fills out the survey exactly once, and fills it out according to the instructions. Your goal is to write a program to find an assignment of students to teams that minimizes the total amount of work the course staff needs to do, subject to the constraint that no team may have more than 3 students. Your program should take as input a text file that contains each student's response to these questions on a single line, separated by spaces. For example, a sample file might look like:

```
djcran djcran-vkvats-nthakurd sahmaini
sahmaini sahmaini _
sulagaop sulagaop-xxx-xxx _
fanjun fanjun-xxx nthakurd
nthakurd nthakurd djcran,fanjun
vkvats vkvats-sahmaini _
```

where the underscore character (_) indicates an empty value. Your program should be run like this:

```
python3 ./assign.py [input-file] [k] [m] [n]
```

where k, m, and n are values for the parameters mentioned above, and the output of your program should be a list of group assignments, one group per line, with user names separated by dashes followed by the total time requirement for the instructors, e.g.:

```
[djcran@tank part3]$ python3 ./assign.py input-file 30 20 10
djcran-vkvats-nthakurd
sahmaini
sulagaop-fanjun
123
```

(In this example, the cost of 123 was calculated as $k \times 3 + 1 \times 3 + m \times 1 + n \times 1 = 30 \times 3 + 1 \times 3 + 20 \times 1 + 10 \times 1$: there are 3 groups, each requiring 30 minutes of grading, three people (sulagaop, nthakurd, and vkvats) didn't get the requested number of teammates, one person (nthakurd) had to work with someone they requested not to work with (djcran), and one person (vkvats) didn't get to work with a person they requested (sahmaini).)

Hint: It may not always be possible to find the actual best solution in a reasonable amount of time, and "reasonable amount of time" may differ from one user to the next. Our grading program will eventually exit your program if it takes too long. One trick is that since our grading program only looks at the last solution that you output, your program can output multiple solutions. This may be useful if your approach can quickly produce an estimate of the solution, and then as it performs more computation, finds better and betters solutions.

What to turn in

Turn in the three 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. **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 the three problems, instead of having to write each one from scratch.