CS B551 - Assignment 1: Searching

Spring 2021
Due: Friday March 5, 11:59:59PM Eastern (New York) time
Late submissions accepted until Sunday March 7 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 the SICE Linux servers.
5. **IMPORTANT:** In addition to testing your programs on your own, test your code on one of the SICE Linux systems such as burrow.sice.indiana.edu using test scripts that we will provide. We will post more details on the test scripts on Q&A Community.

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.

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-sp2021`. 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-sp2021/`*`your-repo-name`*`-a1`

If that doesn't work, instead try:

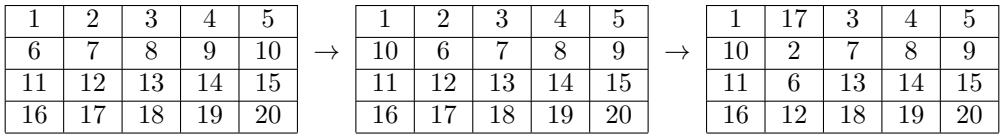
`git clone https://github.iu.edu/cs-b551-sp2021/`*`your-repo-name`*`-a1`

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 2021 Puzzle

Consider the 2021 puzzle, which is a lot like the 15-puzzle we talked about in class, but: (1) it's played on a 4x5 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." However, each given row or column can only be slid in a certain way: the first and third rows can only be slid left; the second and fourth rows can only be slid right; the first, third, and fifth columns can only be slid up; and the second and fourth columns can only be slid down.

For example, here is a sequence of two 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 solver2021.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, each of which 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.)

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 write code that will be able to quickly solve all puzzles.

In addition to doing your own testing, it is important that you test your program on the SICE Linux servers to ensure that we will be able to run it and grade it accurately. We will provide test scripts in Q&A Community at least a week before the deadline.

Part 2: Road trip!

It's not too early 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.
  - `safe` tries to find the safest route — the one that minimizes the probability of having an accident. Assume that the rate of accidents is 1 per million miles driven for the U.S. Interstate Highway

System, and 2 per million miles drives for all other roads. You can identify an Interstate Highway because it will have an "I-" in the name. Assume that the probability of having an accident on any given route is thus one-millionth times the number of Interstate miles plus two-millionth times the number of non-Interstate miles.[1]

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, and expected number of accidents for that route. 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" 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 the SICE Linux servers to ensure that we will be able to run it and grade it accurately. We will provide test scripts in Q&A Community at least a week before the deadline.

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).

[1] Note that this is an approximation, as you'll soon see in the probability lectures; you might suspect that something is wrong by the fact that this probability will go above 1.0 if you drive a few million miles. The true probability of having at least one accident in m miles if the probability of an accident in any given mile is p is $1 - (1 - p)^m$, but our approximation is good for reasonable trip lengths (up to a few thousand miles at least).

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 the number of complaints.

- Each student who requested a specific group size and was assigned to a different group size will send a complaint email.
- Each student who is not assigned to someone they requested sends a complaint email. 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) sends **two** complaint emails. 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 number of complaint email is equal to the sum of these three components (e.g., if a student has multiple complaints, they'll send multiple emails). 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 number of complaints. 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.

We have provided skeleton code to get you started, which can be run like:

```
python3 ./assign.py [input-file]
```

Your job is to complete the solver() function. The function should return the final groups (each named according to the students in the group, separated by hyphens), and the total cost (number of complaints). For example, one assignment for the above file could be:

```
["djcran-vkvats-nthakurd", "sahmaini", "sulagaop-fanjun"]
```

which has a cost of 6 (because three people (sulagaop, nthakurd, and vkvats) didn't get the requested number of teammates (3 complaints), one person (nthakurd) had to work with someone they requested not to work with (djcran) (so they sent 2 complaints total), and one person (vkvats) didn't get to work with a person they requested (sahmaini) (1 complaint).)

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 problem to the next. Our grading program will eventually exit your program if it takes too long. Your program is thus allowed to generate multiple solutions, which 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. You'll call yield() each time you have found an answer — see skeleton code for details.

In addition to doing your own testing, it is important that you test your program on the SICE Linux servers to ensure that we will be able to run it and grade it accurately. We will provide

test scripts in Q&A Community at least a week before the deadline.

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.