CS B551 - Assignment 1: Searching

Fall 2017
Due: Sunday October 1, 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 3 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 three of you personally designed and wrote. You may not share written answers or code with any other students except your own teammates, nor may you possess code written by another student who is not your teammates, either in whole or in part, regardless of format.

What to do. The assignment requires you to write programs in Python. You may import standard Python modules for routines not related to AI, such as basic sorting algorithms and 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 the final version you submit must work on the CS Linux machines.

For each problem, please write a detailed comment 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, 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.

You'll submit your code via GitHub. We strongly recommend using GitHub as you work on the assignment, pushing your code to the cloud frequently. Among other advantages, this: (1) makes it easier to collaborate on a shared project with your group, (2) prevents losing your code from hard disk crash, accidental deletion, etc., (3) makes it possible for the AIs to look at your code if you need help, (4) makes it possible to retrieve previous versions of your code, which can be crucial for successful debugging, and (5) helps document your contribution to the team project (since Git records who wrote which code). If you have not used IU GitHub before, instructions for getting started with git are available on 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-fa2017`. 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 git@github.iu.edu:cs-b551-fa2017/`*`your-repo-name`*`-a1`

   If that doesn't work, instead try:

   `git clone https://github.iu.edu/cs-b551-fa2017/`*`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.)

1. Besides baseball, McDonald's, and reality TV, few things are as canonically American as hopping in the car for an old-fashioned road trip. 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:

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

   where:

   - `start-city` and `end-city` are the cities we need a route between.
   - `routing-algorithm` is one of:
     - `bfs` uses breadth-first search (which ignores edge weights in the state graph)
     - `uniform` is uniform cost search (the variant of bfs that takes edge weights into consideration)
     - `dfs` uses depth-first search
     - `astar` uses A* search, with a suitable heuristic function
   - `cost-function` is one of:
     - `segments` tries to find a route with the fewest number of "turns" (i.e. edges of the graph)
     - `distance` tries to find a route with the shortest total distance
     - `time` tries to find the fastest route, for a car that always travels at the speed limit

   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? (3) Which algorithm requires the least memory, and by how much, according to your experiments? (4) Which heuristic function(s) did you use, how good is it, and how might you make it/them better?

   Extra credit. For some extra credit, implement two additional cost-functions: `longtour` should try to find the longest distance driving path between the two cities that does not visit the same city twice, and `statetour` should try to find a route with the shortest total distance that includes passing through at least one city in each of the 48 contiguous U.S. states.

2. The course staff of a certain Computer Science course randomly assigns students to teams for their first assignment of the semester to help force people to get used to the real-world situation of working on diverse groups of teams of programmers. But since they are not completely heartless, the instructors allow students to give preferences on their teams for future assignments. In particular, each student is sent an electronic survey and asked to give answers to four questions:

0 - No preference
1 - Team of one
2 - Team of two
3 - Team of three

   (a) What is your user ID?
   (b) Would you prefer to work alone, in a team of two, in a team of three, or do you not have a preference? Please enter 1, 2, 3, or 0, respectively.
   (c) Which student(s) would you prefer to work with? Please list their user IDs separated by commas, or leave this box empty if you have no preference.
   (d) Which students would you prefer not to work with? Please list their IDs, separated by commas.

You can assume every student fills out the survey exactly once. Ideally, student preferences would be compatible with each other so that the group assignments would make everyone happy, but inevitably this is not possible because of conflicting preferences. So instead, being selfish, the course staff would like to choose the group assignments that minimize the total amount of work they'll have to do. They estimate that:

This will happen only if the student is assigned to different group size

- 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 after class, taking 1 minute of the instructor's time.
- Each student who is not assigned to someone they requested will send 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 4 above) will request a meeting with the instructor to complain, and each meeting will last m minutes. If a student requested not to work with two specific students and is assigned to a group with both of them, then they will request 2 meetings.
  HE

The total time spent by the course staff is equal to the sum of these components. 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 3 zehzhang,chen464 kapadia
chen464 1 _ _
fan6 0 chen464 djcran
zehzhang 1 _ kapadia
kapadia 3 zehzhang,fan6 djcran
steflee 0 _ _
```

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

```
./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 spaces followed by the total time requirement for the instructors, e.g.:

```
djcran chen464
kapadia zehzhang fan6
steflee
534
```

INPUT: k=160, m=31, n=10

No of teams: 3 , T1 = k*3 = 160*3 = 480
No of students not assigned as per positive preference: 1+1=2, T2 = n*2 = 10*2 = 20
No of students assigned as negative preference: 1, T3 = m*1 = 31*1 = 31
No of students who did not get desired team preference: T4 = 1+1+1 = 3

Hence, T_main = T1+T2+T3+T4 = 480+20+31+3 = 534

Total time = k*no_of_teams + l + m*no_students_not_assigned_as_positive_pref + n*no_students_assigned_as_negative_pref

Positive_preference —> People with whom a student wants to work
Negative_preference —> People with whom a student does not want to work

which proposes three groups, with two people, three people, and one person, respectively.

3. Consider a variant of the 15-puzzle, but with the following important change. Instead of sliding a single tile from one cell into an empty cell, in this variant, either one, two, or three tiles may be slid left, right, up or down in a single move. For example, for a puzzle in this configuration:

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

the valid successors are:

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

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

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

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

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

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

The goal 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 solver16.py that finds a solution to this problem efficiently using A* search. Your program should run on the command line like:

```
./solver16.py [input-board-filename]
```

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

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

where 0 indicates the empty position. The program can output whatever you'd like, except that the last line of output should be a 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, followed by 1, 2, or 3 indicating the number of tiles to move, followed by a row or column number (indexed beginning at 1). For instance, the six successors shown above would correspond to the following six moves (with respect to the initial board state):

```
R13 R23 L13 D23 D13 U13
```

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 SOIC Linux computers.** We will provide a test program soon to help you check this.