CSC 665: Artificial Intelligence
Homework 1

By turning in this assignment, I agree to abide by SFSU's academic integrity code and declare that all of my solutions are my own work.

1 Grid City

Consider an infinite city consisting of locations (x, y) where x, y are integers. From each location (x, y), you can go left, right, up, or down. You start at $(0, 0)$ and want to go to (m, n), where $m, n \geq 0$. We can model this as a search problem as follows:

- $s_0 = (0, 0)$
- $\text{Actions}(s) = \{(+1, 0), (-1, 0), (0, +1), (0, -1)\}$
- $\text{Succ}(s, a) = s + a$
- $\text{Cost}((x, y), a) = 1 + \max(x, 0)$ (it is more expensive as you go further to the right)
- $\text{IsEnd}(s) = \begin{cases} \text{True} & \text{if } s = (m, n) \\ \text{False} & \text{otherwise} \end{cases}$

a. (6 points) What is the minimum cost of reaching location (m, n) (note that $m, n \geq 0$) starting from location $(0, 0)$ in the above city? Describe one possible path achieving the minimum cost. Is it unique (i.e., are there multiple paths that achieve the minimum cost)?

b. (6 points) True or false (and explain): UCS will never terminate on this problem because the number of states is infinite.

c. (6 points) True or false (and explain): UCS will return the minimum cost path and explore only locations between $(0, 0)$ and (m, n); that is, locations (x, y) such that $0 \leq x \leq m$ and $0 \leq y \leq n$.

d. (6 points) True or false (and explain): UCS will return the minimum cost path and explore only locations whose path costs are strictly less than the minimum cost from $(0, 0)$ to (m, n).

Now consider running UCS on an arbitrary graph. In other words, forget about the grid city for the following questions.

e. (6 points) True or false (and explain): If you add an edge between two nodes, the cost of the min-cost path cannot go up.

f. (6 points) True or false (and explain): If you make the cost of an action from some state small enough (possibly negative), you can guarantee that that action will show up in the minimum cost path.

g. (6 points) True or false (and explain): If you increase the cost of every action by 1, the minimum cost path does not change (even though its cost does).

2 Six degrees

(20 points) According to the Six Degrees of Kevin Bacon, anyone in the Hollywood film industry can be connected to Kevin Bacon within six steps, where each step consists of finding a film that two actors both starred in.

In this problem, we're interested in finding the shortest path between any two actors by choosing a sequence of movies that connects them. For example, the shortest path between Jennifer Lawrence and Tom Hanks is 2: Jennifer Lawrence is connected to Kevin Bacon by both starring in "X-Men: First Class," and Kevin Bacon is connected to Tom Hanks by both starring in "Apollo 13."

We can model this as a search problem:

- states are actors
- actions are movies which take us from one actor to another (it's true that a movie can take us to multiple different actors, but that's okay for this problem)
- the start state and goal state are defined by the two people we're trying to connect

The `actors` directory contains two sets of CSV data files: one set in the `large` directory and one set in the `small` directory. Each contains files with the same names and the same structure, but `small` is a much smaller dataset for ease of testing and experimentation.

Each dataset consists of three CSV files. A CSV file, if unfamiliar, is just a way of organizing data in a text-based format: each row corresponds to one data entry, with commas in the row separating the values for that entry.

Open up `small/people.csv`. You'll see that each person has a unique ID, corresponding with their ID in IMDb's database. They also have a name and a birth year.

Next, open up `small/movies.csv`. You'll see here that each movie also has a unique ID, in addition to a title and the year in which the movie was released.

Now, open up `small/stars.csv`. This file establishes a relationship between the people in `people.csv` and the movies in `movies.csv`. Each row is a pair of a `person_id` value and `movie_id` value. The first row (ignoring the header), for example, states that the person with ID 102 starred in the movie with ID 104257. Checking that against `people.csv` and `movies.csv`, you'll find that this line is saying that Kevin Bacon starred in the movie "A Few Good Men."

Finally, take a look at `degrees.py`. At the top, several data structures are defined to store information from the CSV files. The `names` dictionary is a way to look up a person by their name: it maps names to a set of corresponding IDs (because it's possible that multiple actors have the same name). The `people` dictionary maps each person's ID to another dictionary with values for the person's name, birth year, and the set of all the movies they have starred in. And the `movies` dictionary maps each movie's ID to another dictionary with values for that movie's title, release year, and the set of all the movie's stars. The `load_data` function loads data from the CSV files into these data structures.

The main function in this program first loads data into memory (the directory from which the data is loaded can be specified by a command-line argument). Then, the function prompts the user to type in two names. The `person_id_for_name` function retrieves the ID for any person (and handles prompting the user to clarify, in the event that multiple people have the same name). The function then calls the `shortest_path` function to compute the shortest path between the two people, and prints out the path. The `shortest_path` function, however, is left unimplemented. That's where you come in!

Complete the implementation of the `shortest_path` function such that it returns the shortest path from the person with ID `source` to the person with the ID `target`. Assuming there is a path from the source to the target, your function should return a list, where each list item is the next `(movie_id, person_id)` pair in the path from the source to the target. Each pair should be a tuple of two strings.

For example, if the return value of `shortest_path` were `[(1, 2), (3, 4)]`, that would mean that the source starred in movie 1 with person 2, person 2 starred in movie 3 with person 4, and person 4 is the target. If there are multiple paths of minimum length from the source to the target, your function can return any of them. If there is no possible path between two actors, your function should return `None`.

You may call the `neighbors_for_person` function, which accepts a person's ID as input, and returns a set of `(movie_id, person_id)` pairs for all people who starred in a movie with a given person. You should not modify anything else in the file other than the `shortest_path` function, though you may write additional functions and/or import other Python standard library modules.

3 Tic-tac-toe

In this problem, you will use minimax to write a program that plays tic-tac-toe optimally.

To get started, run `pip3 install -r requirements.txt` in the `tictactoe` directory to install the required Python package.

There are two main files in this project: `runner.py` and `tictactoe.py`. `tictactoe.py` contains all of the logic for playing the game, and for making optimal moves. `runner.py` has been implemented for you, and contains all of the code to run the graphical interface for the game. Once you've completed all the required functions in `tictactoe.py`, you should be able to run `python runner.py` to play against the AI! *Please do not modify `runner.py`. On Canvas, you will only submit `tictactoe.py`.*

a. (3 points) Implement `player`.

b. (3 points) Implement `actions`.

c. (6 points) Implement `succ`.

d. (6 points) Implement `winner`.

e. (3 points) Implement `terminal`.

f. (3 points) Implement `utility`.

g. (14 points) Implement `minimax`.

Once all functions are implemented correctly, you should be able to run `python runner.py` and play against the AI. Since tic-tac-toe is a tie given optimal play by both sides, you should never be able to beat the AI (though if you don't play optimally as well, it may beat you!).

Submission

Submission is done on Canvas. You should submit three files: one containing your solutions to the written problems, and one for each of the two coding problems.

- Submit your written solutions in a single PDF file. Make sure to clearly indicate the number and letter of the problem corresponding to each solution. It is okay to hand-write your solutions and then scan them into a PDF, but *only if your handwriting is legible.*
- Submit your coding solutions in the provided `.py` files. For problem 2 you should submit `degrees.py`, and for problem 3 you should submit `tictactoe.py`. Do not modify the names of these files after downloading them from the course website.