CSCI 4041, Fall 2018, Programming Assignment 10

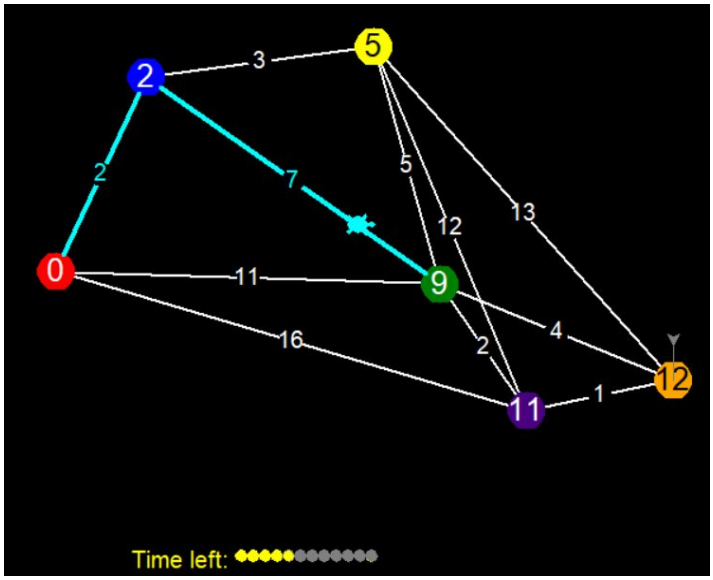
Due Tuesday, 11/13/18, 10:30 AM (submission link on Canvas)

This is not a collaborative assignment; you must design, implement and test the solution(s) on your own. You may not consult or discuss the solution with anyone other than the course instructor or TAs. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class. Obtaining or sharing solutions to any programming assignment for this class is considered academic misconduct. If you are not sure what this means, consult the class syllabus or discuss it with the course instructor.

(Note: the first page is entirely fluff information. If you don't like ridiculous stories used to justify doing the problem, you're free to skip to page 2 where we start talking about implementation)

You are the captain of the Cyan Turtle, the most reflective spaceship ever built. Your home system, Orange Beta, is being attacked by a mysterious alien battleship. They seem to be using a beam weapon that warps entire planets into another dimension. You have one chance: you need to place your ship in the path of the beam when it fires, and hope that your hyper-reflective hull bounces the beam back at the attacker.

The problem is that you're currently hanging out in the Red Sigma system, which is on the other side of the galaxy. There is at least one route from Red Sigma to Orange Beta through the hyperspace lanes, which means that it might still be possible to reach your home before the battleship's beam finishes charging and fires, but your navigational computer is broken, so you're not sure what the fastest route to take is. Hyperspace lanes vary wildly in their required travel time, and the time required to make a hyperspace jump is not necessarily correlated to linear distance between the source and destination. So, you'll need to use Dijkstra's algorithm on your Star Map to find the fastest route from Red Sigma to Orange Beta.



You must find the fastest path from the Red star to the Orange star.

Download the PA10.py template from the course website.

The template contains the Star class (essentially a single vertex for the purposes of Dijkstra's algorithm), which stores information on each Star system like

- their color (visual representation only, not needed for the algorithm)
- index number (a value between 0 and n-1, where n is the number of nodes: this is used for indexing into the hyperspace jump time matrix/lists)
- a list of all stars connected to this one through a hyperspace lane (an adjacency list in no particular order)
- a jump times list (an integer list of the edge weights between this node and every other node, in order of index number: essentially this node's row in the edge weight matrix; unlike the adjacency list this one will include an entry (infinity) for nodes not adjacent to this one)
- a **private** instance variable __dist, which represents the best distance estimate used in Dijkstra's algorithm (.d in the textbook). This variable is private to force you to use the getter and setter methods get_dist() and set_dist() to alter it, which will adjust the number drawn on top of the node within the Star Map.
- a .prev pointer (.pi in the textbook) representing the previous star in the shortest path from the start node (the red star) to this star.

There are also functions within the template that you may use for implementing a min-priority-queue, and some test cases.

You must implement the `hyperspace_jump_route` function, which takes as input a list of Star objects (essentially a list of vertices), and a matrix of integers representing the jump times (edge weights) between any pair of Stars, and uses Dijkstra's algorithm to compute the shortest path from the Red star to the Orange star. The Red star (the source) will always be the first element of the list (star_list[0]), and the Orange star will always be the second element of the list (star_list[1]). Unlike the textbook version of Dijkstra's algorithm, this function must actually return the final path from the start node to the goal node, inclusive.

Requirements:

- You must download the template file PA10.py and edit the `hyperspace_jump_route` function. You can create your own helper functions, but don't edit the code beyond the "DO NOT EDIT" line. There is one exception to this: you are permitted to switch the flag on turtle.tracer(1) to a 0 if you want to speed up testing.
- You must complete the functionality of `hyperspace_jump_route` by using Dijkstra's algorithm: there are no hyperspace jumps that require negative time, and a Bellman-Ford implementation isn't fast enough to save your planet.

- Code has been included to implement the min priority queue data structure for Dijkstra's algorithm, but you are not required to use it: feel free to edit them or implement your own.
- Your program must run without errors on the version of Python installed on the CSELabs machines, Python 3.5.2. (if you're testing this on CSELabs, you need to type python3 or idle3 instead of python or idle to start the correct version from the terminal)
- You are not permitted to use the input() function as this will break the grading script, or the built in .sort() or sorted() functions.
- You must implement Dijkstra's algorithm, as found in Chapter 24.3 of the textbook. Any other algorithm will receive very little credit.
- This assignment will be graded automatically based on the number of test cases your program passes. There will be several secret test cases in addition to the ones included in the template to ensure you're not hard-coding in the solutions.
- This program will only run test cases until you fail one, avoiding the problem of having to scroll through test output to find the one broken test case.
- The grading breakdown for this assignment is as follows:
    - 30%: File runs without syntax errors
    - 70%: Passing test cases without breaking any requirements.
- The unedited template file already runs without syntax errors. This means that if your program causes syntax errors, you will get a better score by just submitting the original template unedited.
- Submit your edited PA10.py file to the Programming Assignment 10 link on Canvas before 10:30 AM on 11/13/18. No credit will be given for late submissions.