Scenario:

You are a programmer at a software company.  The company's product is a software program that offers travel recommendations to customers.  The company has a successful tool for recommending hotel accommodations and is now trying to roll out a tool that will recommend flight arrangements.  For the flight recommendation tool, here are the basic elements:

* airport:  An airport is some actual airport in the real world, typically identified by its airport code (e.g., LAX for Los Angeles International Airport).
* flight:  A flight is a flight from one airport to another.  Each flight has an associated cost.  A flight goes from airport A to airport B, but not necessarily vice versa.
* route:  A route is a sequence of flights to get from a departure airport (i.e., starting) to an arrival airport (i.e., ending), such as:  (airport A -> airport B) then (airport B -> airport C) for departure = airport A, and arrival = airport C

So far, things have not been going well with the flight recommendation tool.  A previous programmer has started the project and got some functionality coded.  In particular, the previous programmer has written the following tools:

(1) A function that determines if a customer can fly from a departure (i.e., starting) airport to an arrival (i.e., ending) airport.  This checks whether there is any possible route from the departure airport to the arrival airport.  This is the function can\_I\_fly\_there().

(2) A function that determines the cheapest route from a departure airport to an arrival airport.  This function finds the route with the lowest composite cost that starts at the departure airport and ends at the arrival airport.  This is the function what\_is\_the\_cheapest\_route.

But the tools (1) and (2) are apparently rather buggy.  And worse still, the previous programmer has completed failed to implement two other important tools:

(3) A function that determines the cheapest route from a departure airport to an arrival airport while not making any layovers in "dirty" airports.  Some airports are just downright gross, and travelers do not want to stop there (unless of course the airport is the departure or arrival airport, in which case the traveler has no choice).  So the function what\_is\_the\_cheapest\_route\_with\_no\_dirty\_airport\_layovers() should find the cheapest route from a departure airport to an arrival airport, but without making any layovers in dirty airports.

(4) A function that determines the route from a departure airport to an arrival airport with the fewest number of layovers.  Travelers hate layovers, so the tool needs to be able to find the route that finds the route that minimizes layovers.  When multiple routes tie for this result (e.g., multiple routes have only one layover, which is the best result), the tie should be broken first by which route has the fewest dirty airport layovers, and (if a tie remains) then by which route is cheapest.

The previous programmer has been fired for gross incompetence.  Now its your turnt to step into the breach and get this tool working.  Your job is to deliver the software programs that implement the functions (1), (2), (3), and (4) free of bugs, and do so by 11:00 PM next Tuesday.

The existing software [is provided hereView in a new window](https://canvas.eee.uci.edu/courses/5731/files/2067246/download?verifier=9IPCXyjtbm1SCm38WHVl7Ct8mTgjejYQKoN3LNi0&wrap=1), and you should implement your solution therein.  Good luck!

Assignment Details:

With the foregoing scenario as the backdrop, here are further details for the assignment.

Because of the nature of this homework assignment, **collaboration is not permitted**.  But you can still ask questions of Kyle and Grady.

There are two parts to the assignment, both of which should be performed in the route\_analyzer module:

(A) Unit test and debug the existing code;

* The existing route\_analyzer module has 10 bugs that the previous programmer accidentally left in the code (i.e., that we purposefully introduced).
* Your job is to unit test the existing code in route\_analyzer to remove as many of the 10 bugs as possible.
* Each bug removed earns 10 points.
* The grading will consist of us running ten unit tests (that capture each bug) against your modified route\_analyzer module.  You will get 10 points for each of the unit tests that pass with your modified route\_analyzer module.
* The bugs can occur anywhere in the route\_analyzer module, and you are responsible for fixing any bugs anywhere in that module.

(B) Write the new functions (3) and (4).

* The existing route\_analyzer module has stubs for functions (3) and (4).
* Your job is to implement these functions.
* You may need to make changes elsewhere in the route\_analyzer module.
* Each function is worth 25 points.
* The grading will consist of our standard approach of running different tests against your new functions.

Thus in total, the homework is worth 10 x 10 + 2 x 25 = 150 points.

Here are some additional instructions for the assignment:

* You should write your unit tests in the hw7\_tester module.
* **THERE WILL BE NO SANITY CHECKER PROVIDED.  YOU ARE WRITING THE SANITY CHECKER IN THE FORM OF UNIT TESTS**.
* You only need to modify the code in the route\_analyzer and hw7\_tester modules.  If you encounter some reason to need to change the code in the edge, graph, or vertex modules, then please let Kyle/Grady know first.
* **BUT**, you cannot change the signature on any of the functions.  That is, all functions that exist in the initial code can be changed internally, but you cannot change any of:  the name, the number/purpose of input parameters, or the return type.  Doing so will render the code inoperable and thus will fail the grading test cases.  (You would essentially be breaking backward compatibility with the existing software inside the company that uses those functions.)
* You can of course add helper functions if you would like.

Here are more details on what you need to submit to Canvas by the deadline:

* You should submit a compressed zip directory containing all of the following modules:  edge, graph, hw7\_tester, route\_analyzer, vertex.
* You should also submit the hw7\_bug\_list file that enumerates the bugs that you believe that you corrected in the route\_analyzer module.  Following the example, there should be a brief statement of the bug (140 characters max) and the name of the unit test (from hw7\_tester module) that demonstrates this bug.

Here are a few notes on what constitutes a bug, as well as what we will test for your functions (3) and (4):

* The code in the route\_analyzer module must conform to the docstrings included therein.  So if the docstring says that a class/function should behave a certain way, and it does not, then that is a bug.
* The 10 bugs are not marginal cases.  That is, if you find a behavior in the code and are not sure if it is correct or not, then it is probably not one of the 10 bugs.  The 10 bugs we are testing for cause rather blatant errors in the code's execution. If you are unsure, run through the code by hand and check if the result you get makes sense.
* We will not be looking for poor performance (e.g., code runs too slow) as a bug, but extremes (e.g., an infinite loop) would naturally count as a bug.

Here is a running list of answers based on student questions:

* If you end up finding more than ten bugs (like I said, there could be some unintentional ones there), then go ahead and document all of them in hw7\_bug\_list file.
* The failure of a function to handle (e.g., raise an exception) on invalid input is not one of the bugs.  For example, if you can can\_I\_fly\_there on an invalid airport code and something bad happens (e.g., unhandled exception), that is not a bug.  While ideally these invalid inputs would be checked and handled, that would really make the scope of possible bugs endless, so I avoided using invalid inputs as any of the bugs.  But because "invalid input" is a rather vague term, here are some examples from student question:
  + Passing an argument to a function that is of the wrong data type, and the function failing to handle that properly is not one of the bugs.
  + Calling can\_I\_fly\_there or any of the route finder functions with airport codes that do not exist, and the function failing to handle that properly is not one of the bugs.
* It is not a bug in this assignment if an iterator does not support nested for loops, or if an iterator changes after the underlying data structure changes.  While those were specific requirements for the last homework, they are not requirements here.
* There were some questions about the code at lines 100-101 in route\_analyzer, the code that raises an exception based on numerical comparison.  I think the meaning of this block will be evident after some analysis and debugging.  But do remember that you are free to change the implementation of any of the functions as you see fit (just not the signature and purpose of the functions), so feel free to delete that code if you want.  For clarity, though:  that code is not in and of itself a bug.
* Hint: [Dijkstra's algorithm for shortest paths (Links to an external site.)Links to an external site.](https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm)