Back-end Architecture Description:

The back-end of our algorithm relies on two overarching functions that collect and the format the solutions as needed. This is searchFlights and sortFlights. Once the overarching function “getFlightSolutions” goes through both of those functions, it takes the List of solutions that are returned, and returns a subset of the solutions, so that only the number specified in the query is returned.

Explanation of Classes:

Query:
Contains all of the data required in a query according to the spec

Flight:
Contains all of the information needed for a flight, e.g. start location, destination, time, airline…

Path:
A flight solution that contains the information of the solution e.g. duration, start location, end location…

It also contains a list of all the flights included in the solution

Graph:
The class that contains all of the cities entered as well as the flights between them

SearchFlights:
Takes in: Query (q) and Graph (g)
Returns: List of Paths

Overview:

This function is the core of the flight tracking system as it’s the one that gets all of the flight solutions that are needed to complete the spec. This is done via a breadth first search. SearchFlights is done by initializing a blank flight solution that starts at the appropriate location and then loads that into the queue that will be used in the breadth first search.

The loop of the breadth first search then takes the first solution out of the queue and then checks it to see if it reaches the destination. Assuming it does, that solution is added to the list of solutions to return. Otherwise, the current solution is passed to a function on the graph: “getFlights”. What this does, is it takes the current solution and returns a list of flights that the current solution can take from its state at that time. Each of those are then iterated through to generate a new set of solutions. One for each flight returned by “getFlights”. Each of those solutions are then put onto the queue so that they are checked in one of the next passes.

The search will continue until the queue is empty so that every possible solution that meets the conditions will be put onto the list returned by SearchFlights.

SortFlights:

Takes in: List of Paths (flightList)  and  Query (query)

Returns: List of Paths

Overview:

This function applies a python sort to a list of flight solutions (Paths) based on the preferences. This is done by looking at the preferences the user input, which are being stored in the query class. Once it has checked for the order of the preferences, it then runs the python sort using a lambda function with the appropriate ordering of preferences.

This is part of standard python modules and works outside the abstraction level of our back-end implementation.