2019 Career Fair Programming Competition

Iddriss Raaj (28002022)

Step 1:
Parse argument from commanline and open both input and output files in memory; get input parameters from input file i.e *source city*, *country* and *destination city*, *country*

Step 2:
Call *optimalRoute* functions and pass input parameter. The *optimalRoutes* acts as a main function in the *functions* module. The *optimalRoute* function makes calls to; *getAirports, createGraph* and *gettRoute* passing in their respected expected parameter.

Step 3:
The *getAirports* function takes in three parameters, *city, country* and *id* respectively. All of which has been initialised to empty. This function basically returns a list of all airports available in the provided *city* and *country* combination or an airport id (*id)*. This is achieved by looping through the airport databse and filtering out airports that satisfies the condition.

Step 4:
Using the source airport and destination airport list, I create dictionary relating the parent nodes to child nodes from the list of routes available by calling the *createGraph* function. I achieved this by partially implementing a Depth-First Search algorithm. In this function I make a call to *getNodes,* this return a list of possible route nodes whiles ensuring that the Airline of each route exist. The *createGraph* will return a dictionary containing all starting points but only one destination point which is used by *getRoute* to get the shortest possible route.

Step 5:
Once I have my dictionary of routes, I call *getRoute* function which takes the dictionary of routes, source aiports and destination airports. I loop through the routes and gradually retrace from the only available destination to get the shortes route to the start. The function then returns the found route which is then used to populate the output file.

Step 6:
Finally, using the shortes route and haversine formular, I loop throug the routes and get their respective airpot to find their longitude and latitude which is then passed to the *haversine* function to get the total distance.

References:
Breath-First Search: https://en.wikipedia.org/wiki/Breadth-first_search

Depth-First Search: https://en.wikipedia.org/wiki/Depth-first_search

BFS and DFS implementations: https://eddmann.com/posts/depth-first-search-and-breadth-first-search-in-python/

Haversine : https://en.wikipedia.org/wiki/Versine#Haversine

Graph theory : https://en.wikipedia.org/wiki/Graph_theory