Documentation

Eduardo Parkinson de Castro
OOL - Dublin Institute of Technology
D17126897

Overview

The programs are written in python 3.0. It is supposed to calculate the most optimum(least expensive) and least optimum(most expensive) order of trips given 6 airports. The program receives input from the user. The first input is the home airport which is then applied again to the algorithm because we assume the pilot always returns home. The following csv files are loaded by the program:

1. aircraftcsv.csv
2. airport.csv
3. countrycurrency.csv
4. currencyRates.csv

The program also makes use of a number of imports. The imports used in the program are the following: csv, os, itertools, math and numpy.

The cheapest Route is obtained by calculating 5!=120 and then iterating over the possible permutations. The program takes the users input for different airports and then returns all possible routes in that permutation. In order to do this we create a list of the airports and then run the program itertools over that list.

```
trips_possible_1 = list(itertools.permutations([home,airport_2,airport_3,airport_4,airport_5], 5))
```

This creates a new list that is a tuple with all the possible variations for trips for the airports. The program assumes the first airport code is the same as the last airport, hence the pilot always returns home. The next step is to calculate the cost of each permutation by multiplying the distance of the trip by its cost. By default we are using EU GDP and USD to calculate the different costs.

The next step is to organise the trip cost and the airport variation into a final cost tuple that contains all possible trips and their associated total costs.

My total_trip_costs array became: total_trip_costs.append([total costs][tuple_values]).
My tuple_values represent all possible permutations obtained from all permutations found in my trips_possible_1 list so by putting them in an array I can map the total_cost with the associated flight itinerary permutation.

In order to find the minimum cost and the maximum cost we used numpy. With numpy sorted I was able to sort the first value of the array from least to greatest. The last value in the index of the array was therefore my cheapest route permutation and the first was my most expensive.

Files

currency.py - This files contains the class Currency that is meant to deal with the currency.csv file. It takes the currency's in the csv file and places them in a dictionary where they currency code is the key. It has a createInstance method that creates an object for the currency in the csv file and places it into a dictionary that contains the currency's code, name, euro_to_currency value and currency_to_euro value. It also has a method used to find that currency's conversion rate (euro_to_currency) value.

airport.py - This files contains the class Airport that is meant to deal with the airportcsv.csv file and the elements in the file. It takes the airports in the csv file and places them in a dictionary where the key is the airport code. It has a createInstance method that creates an object for a specific code in the dictionary and then finds all associated attributes for that airport.

CountryCurrency - This files contains the class CountryCurrency that is meant to deal with the country currency.csv file and the elements in the file. It takes the country currency elements in the csv file and places them in a dictionary where its alphabetic code is the key. It also disregards any redundant elements in csv file and creates a dictionary with only currency name, alphabetical code and its country. It has a createInstace method that creates objects of country currency from the csv file and gets its other attributes. It also has a findCurrencyCode method meant to return that currency's code so that its coversionrate can be located.

AirportAtlas.py - This is the main file in the program. It contains all the class AirportAtlas that deals with with airportcsv.csv file and doing all the calculations. It has a method for getting airport distances between two airports and another for getting the total distance between 4 airports that was used to test the calculations over time. The possibletrips method deals with finding the different permutations, getting the airport codes and sending them to the calculate distance method which gets the airport's coordiantes for latitude and longitude and send them to the calculate_distance method which then calculates the distances the airports are from each other the over the arc of the earth (which we assume to be 6371).

AirportAtlasTest.py - This is where I test the functionalities of AirportAtlas. It takes the input from the user running the program for the airport code and capitalises them then runs the values against the file.

CountryCurrencyTester - this file contain the tests used to check CountryCurrency objects its respective methods.

CurrencyTester - this file contains the tests used to check currency objects and its respective methods.

I left out the features on the specific aircrafts and fuel capacity because I did not manage to make it. I also did not manage to create a GUI for the project so unfortunately all interactions with the user has be done with the command line (by running AirportAtlasTest.py). Many functions of the program (like showing price specific distances for trips) have also been commented out to facilitate reading of output. Comments in the files are often for my own (and the readers) understanding of what is going on. Other times I comment out my code so I can go back to it and re-visit my attempt or do more research on the topic.