ICP Project 2

Richard Quayson

Department of Computer Science and Information Systems, Ashesi University

CS313_B: Reflection

November 29, 2022

The goal of this project was to write a program that, given a source city and country and a destination city and country, retrieves the shortest path between the airports in the given locations. The project provided data on Airports, Airlines and Routes from OpenFlights. My partner on the project, Paa Kwasi, created classes that provided functionalities for creating objects out of the Airport, Airline and Route data using the read method I implemented. The read method is a method of the ReadWriteFile class and nests the getline method to read CSV files by splitting the lines and, later, the columns. With his help, we handled the edge cases in Airport object creation where there were extra commas in the airport name and city columns of the Airport CSV file. With these classes, I created methods that made use of the CreateObjects class to populate the data read into an unordered map. The unordered map for the airport and airline data has the IDs as key and the entire object as the value. For the route, the key is the id of the source airport, and the value is all Route objects that have that airport id as their source airport id.

Next, I created the AirRoutePlanning class, which takes the source airport id, the destination airport id and the route data as arguments. This class creates objects to be used by the search algorithm. From there, I created the Node class that represents a given airport, the airport from where we got to this airport (parent), and the path cost representing the distance between the root (start airport) and the current airport. This is equally to be used by the search algorithm.

Subsequently, I created the search algorithm class, which defines a uniform cost search method. This partially implements the uniform cost search algorithm. The data structure for the explored states was an unordered set. However, that of the frontier was a double-ended queue. I eventually used a deque instead of a priority queue because of issues I had with traversing the priority queue. This equally made me partially implement the UCS algorithm since determining the item with the same state and a lower path cost on the frontier was challenging. I also created the Solution class, which takes the action sequence from the search and the path cost as instance variables and creates objects of them. Once the search finds a valid path, it creates a Solution object and adds it to the validPaths (a static variable that takes Solution objects and stores them in a vector) in the helper class. Afterwards, the program gets the Solution object with the lowest path cost from ValidPaths, creates a solution string, and writes it to the output file. Overall, this project was quite challenging, not because of how I wanted to solve the problem but because of my inadequate knowledge of the data structures in C++ and also the limited methods they provide. Current me fancies Java and Python over C++; however, would future me prefer C++ over Java and Python? Maybe, maybe not.