Overview

This project was written in Java using SDK version 20, Java version “20.0.2,” and IntelliJ IDEA 2023.2.5, the IDE.

This program will utilize data from two input files to compute the overall flight path and the total flight cost between a source city and a destination city. The depth-first search algorithm will be used to iteratively search the most efficient path between a source city and a destination city. The algorithm will also be able to backtrack and find the most efficient path from a destination city to a source city.

Method Descriptions

readFlightData: This method will read in a text file that includes the flight details such as source city, destination city, flight cost, and flight duration. It will process each data row and create the flight by calling the method addFlight.

readFlightRequest: This method will read a text file that requests information on the flight path between two given cities and how to sort the result based on either cost or time. After reading, the method will process each data row and call the DFS algorithm to execute the flight computation to find the best paths according to a specific sorting preference. The method will write the result to an output file.

addFlight: This method creates a flight path between two cities. It checks if a city is not already in the list of cities and creates a node. This method adds a bidirectional path between the two cities.

findCity: This method searches for a city in a list of cities given a specified name. It will return the city’s node; otherwise, it will be null if not found.

cityVisited: This method checks whether the current city being traversed has been visited by DFS yet. It works by backtracking through the path to check for the city.

performDFS: This method performs a depth-first search to find the most efficient path between a source city and a destination city. The algorithm uses a stack to keep track of all the paths to explore. If the stack is empty, then the algorithm has explored all possible paths.

Class Descriptions

FlightPlanner: This class reads and processes flight data and performs a depth-first search to compute the most efficient flight path between two cities.

SortPath: This class will use a Comparator interface to sort the flight paths based on a preference. The class will either sort the paths by cost or by time.

CityNode: This class holds the flight details for each city. Each node includes the city name, the flight cost, and the flight duration.

CityPath: This class represents the direct flight path between two cities. The information stored is the destination city, the flight cost, and the flight duration.

PathNode: This class keeps track of the flight journey as it traverses through the cities.