Rushi Samirbhai Patel rs397441@dal.ca
B00886157

Assignment 3

Test Cases:

Input Validations

1. boolean addCity()
    - City name is null or empty.
    - City name is entered in a format other than String.
    - City name has special characters.
    - Test Required is null or empty.
    - Test Required is entered in a format other than boolean.
    - Time to test has value in negative.
    - Time to test is entered in a format other than int.
    - Nightly hotel cost is negative.
    - Nightly hotel cost is entered in a format other than int.

2. boolean addFlight()
    - Start city is null or empty.
    - Start city is entered in a format other than String.
    - Start city has special characters.
    - Destination city is null or empty.
    - Destination city is entered in a format other than String.
    - Destination city has special characters.
    - Flight time has value in negative.
    - Flight time is entered in a format other than int.
    - Flight cost has value in negative.
    - Flight cost is entered in a format other than int.

3. boolean addTrain()
    - Start city is null or empty.
    - Start city is entered in a format other than String.
    - Start city has special characters.
    - Destination city is null or empty.
    - Destination city is entered in a format other than String.
    - Destination city has special characters.
    - Train time has value in negative.
    - Train time is entered in a format other than int.
    - Train cost has value in negative.
    - Train cost is entered in a format other than int.

4. List<String> planTrip
    - Start city is null or empty.
    - Start city is entered in a format other than String.
    - Destination city is null or empty.

- Destination city is entered in a format other than String.
- Is Vaccinated is entered in a format other than boolean.
- Cost importance has value in negative.
- Cost importance is entered in a format other than int.
- Travel time importance has value in negative.
- Travel time importance is entered in a format other than int.
- Travel hop importance has value in negative.
- Travel hop importance is entered in a format other than int.

Boundary Cases

1. boolean addCity()
    - City name has only one character.
    - Time to test has value 0.
    - Time to test has value 1.
    - Nightly hotel cost has value 0.
    - Nightly hotel cost has value 1.

2. boolean addFlight()
    - State city has only one character.
    - Destination city has only one character.
    - Flight time has value 0.
    - Flight time has value 1.
    - Flight cost has value 0.
    - Flight cost has value 1.

3. boolean addTrain()
    - State city has only one character.
    - Destination city has only one character.
    - Start city is the same as the destination city.
    - Train time has value 0.
    - Train time has value 1.
    - Train cost has value 0.
    - Train cost has value 1.
    - Adding a train when no cities were added.
    - Adding a train when only one city is added.
    - Adding a train when the city is not added.
    - Adding a train when the destination city is not added.
    - Adding a train when the start city and destination city are added.
    - Adding a train when multiple cities are added.

4. List<String> planTrip
    - State city has only one character.
    - Destination city has only one character.

- Start city is the same as the destination city.
- Planning a trip when no cities were added.
- Planning a trip when only one city is added.
- Planning a trip when the city is not added.
- Planning a trip when the destination city is not added.
- Planning a trip when the start city and destination city are added.
- Planning a trip when multiple cities are added.
- Cost importance has value 0.
- Cost importance has value 1.
- Travel time importance has value 0.
- Travel time importance has a value 1
- Travel hop importance has value 0.
- Travel hop importance has a value 1.

Control Flow Cases

1. boolean addCity()
    - Duplicate city name.
    - Add a city when there are no cities added.
    - Add a city when there is one city added.
    - Add a city when there are many cities added.
    - Add a city where a test is required.
    - Add a city where a test is not required.
    - Time to test has value more than 1.
    - Nightly hotel cost has a value more than 1.

2. boolean addFlight()
    - Adding a flight when no cities were added.
    - Adding a flight when only one city is added.
    - Adding a flight when the start city is not added.
    - Adding a flight when the destination city is not added.
    - Adding a flight when the start city and destination city are added.
    - Adding a flight when multiple cities are added.
    - Start city is the same as the destination city.

3. boolean addTrain()
    - Adding a train when no cities were added.
    - Adding a train when only one city is added.
    - Adding a train when the start city is not added.
    - Adding a train when the destination city is not added.
    - Adding a train when the start city and destination city are added.
    - Adding a train when multiple cities are added.
    - Start city is the same as the destination city.

4. List<String> planTrip

- Planning a trip when no cities were added.
- Planning a trip when only one city is added.
- Planning a trip when the start city is not added.
- Planning a trip when the destination city is not added.
- Planning a trip when the start city and destination city are added.
- Planning a trip when multiple cities are added.
- Start city is the same as the destination city.
- Planning a trip when the cost importance has value more than 1.
- Planning a trip when the time importance has value more than 1.
- Planning a trip when the hop importance has value more than 1.
- Planning a trip when traveller is vaccinated.
- Planning a trip when traveller is not vaccinated.

Data Flow Cases

1. List<String> planTrip
    - Planning a trip when all three cost importance, time importance and hop importance are greater than or equal to one and all have the same values.
    - Planning a trip when cost importance is greater than time importance but is equal to hop importance.
    - Planning a trip when cost importance is greater than the hop importance but is equal to time importance.
    - Planning a trip when cost importance is greater than both hop importance and time importance.
    - Planning a trip when time importance is greater than cost importance but is equal to hop importance.
    - Planning a trip when time importance is greater than hop importance but is equal to cost importance.
    - Planning a trip when time importance is greater than both hop importance and cost importance.
    - Planning a trip when hop importance is greater than time importance but is equal to cost importance.
    - Planning a trip when hop importance is greater than cost importance but is equal to time importance.
    - Planning a trip when hop importance is greater than both hop importance and cost importance.

External Documentation:

Overview:
The travel planner system is to plan a trip during COVID based on various factors related to the ongoing pandemic. The system would expect to add cities in the system before planning a trip and the flights or trains connecting to the cities must also be added before planning a trip. The

system will then calculate the shortest path based on the convenience of the traveller (cost, time and number of hops).

Files and External Data:
I have used 8 java files which are Main.java, Train.java, Flight.java, TravelPlannerImplementation.java, City.java, DijkstrasImplementation.java, TravelAssistant.java, TravelPlanner.java and VertexClass.java.
The main java file will ask the inputs from the traveller (user), and based on the command received would then navigate it to the TravelPlannerImplementation java file. This file is the main implementation of the system where the actual operations take place. I have also created two classes: Train.java and Flight.java for the convenience to store the specific information related to Train and Flights. To develop a graph, I have created one class VertexClass which would also store 'mode of transportation' as flight or train along with other details like start city, destination city, cost and time, which would then create a sort of edge in the graph and the cities are then connected with each other based on the added flight or train. I have not used any external data to implement this system.

Data Structures and their relations:
I have only used List as data structures for my implementation and have further extended the properties of ArrayList to perform my operations.

1. ArrayList<City> cityList : To store the City objects
2. ArrayList<String> cityNamesAddedList: List for all the city names added.
3. ArrayList<Flight> flightList : List for all the flights object added.
4. ArrayList<Train> trainList : List for all the trains object added.
5. ArrayList<VertexClass> edgeWithDetails : List of all the flights or trains added.
6. ArrayList<ArrayList<VertexClass>> adjecencyList: Adjecency list used to denote the graph.
7. ArrayList<String> destinationCityLinkedWithStartCity: List of destination city's names linked with a starting city used to plan trip.
8. ArrayList<String> destinationCityLinkedWithStartCityForBetterRoute//List of destination city's names linked with a starting city used to plan trip for better route instance.
9. ArrayList<VertexClass> list : List to store the vertex class instances.
10. List<String> finalList: Final return list to be used.

Assumptions:

- The value of cost is in the same currency for every operation.
- If start city and destination city are same then would return illegal argument exception

Key Algorithms and Design Elements:
I have added the cities, flights and trains in the beginning in their particular ArrayList of Objects. I have mainly used Dijkstra's Algorithm to find the shortest path from a source city to all other cities. As mentioned in the above sections, I have created an adjacency list when a flight or train is added between two cities. This adjacency list is then converted to an adjacency matrix as I

figured this would be easier to implement in the Dijkstras algorithm using only ArrayList. Once the matrix was created, I used that information to generate a path from the source city to the destination. Furthermore, this path is then used to fetch the details of all the cities from the adjacency list created and returns the path with mode of transportation in a List<String>.

Limitations:

- The system cannot give the shortest distance from source city to destination city while keeping in mind the vaccination status of the traveller. It will assume that the traveller is vaccinated.
- The system can only provide the shortest distance between source and destination cities with respect to cost and time importance. For hop importance, the system can only deduce this if there are less than three edges added in the graph.

References:

Dijkstra's Algorithm
[1] GeeksforGeeks. 2021. Dijsktra's algorithm. [online] Available at: <https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/?ref=lbp> [Accessed 26 October 2021].