Written Report

A. Airplane travel within the United States airports
Goal: Based on the flight information from January 2016, figure out what pathway to get to another airport based on the data given

Dataset: sources: Kaggle
Link: https://www.kaggle.com/datasets/rulyjanuarfachmi/domesticusairflight2016-2018
18 million rows of flight data over three years.

B. Data Processing
How did you load it into Rust?
I load the data into Rust as a CSV file. The process required me to first work with Python to isolate the specific columns I wanted to work with, which were the origin airport, destination airport, and Actual Elapsed Time, which I renamed 'air time'. I also limited the point to only 10,000 flights.

C. Code Structure
Modules
Three modules

1. graph.rs
    a. The file was used to create a hashmap of the hash maps, providing a more structured input for the Dijkstra algorithm.

Key Functions & Types (Structs, Enums, Traits, etc.)
Purpose
Struct - Flightstats stores the flight information for a route, which includes the average and standard deviation as floats, and the count as a usize.

Inputs/Outputs:
It takes the time from the data frame and outputs the count average and the standard deviation of the specific flight route, e.g., JFK to LAX

Key Functions & Types (Structs, Enums, Traits, etc.)
Purpose
A type called a graph that represents a directed weighted graph

Inputs/Outputs:
It takes the airport code and makes the flight stats the edge weights
The type is a nested hashmap.

Key Functions & Types (Structs, Enums, Traits, etc.)
Purpose:
A function called calculate that computes the average and standard deviation.

Inputs/Outputs

- Vector of flight times
- And it outputs the average and the standard deviation as a tuple

Core logic and key components

- Use the total divided by the sum to get the average
- For the standard deviation, I first calculated the variance and then squared it to obtain the standard deviation.

Key Functions & Types (Structs, Enums, Traits, etc.)
Purpose:
Function called build airport that builds the flight graph from the dataframe

Inputs/Outputs

- A dataframe with the origin, destination,n, and time
- The output is the graph with FlightStats as the edges

Core logic and key components

- Parse through the rows and create the flight stats. I then calculated the average and standard deviation of the different values in the different hashmaps.

2. path.rs
    a. The path utilizes Dijkstra's algorithm to determine the shortest path between the airports and the time required to reach the area. Split into its modules to make the algorithm for how to get the graph.

Key Functions & Types (Structs, Enums, Traits, etc.)
Function called fastest_route

Purpose:
Used Dijkstra's algorithm to find the shortest path between two airports

Inputs/Outputs

- Input was the flight graph created in the graph.r,s the start and end airport codes
- The output is the total time, and the pathway needs to be taken or none if the path cannot be found.

Core logic and key components

- pThe function uses binary heaps and reverses it to a minheap for a priority queue, it then finds the shortest known distance and previous nodes, rev tracks the previous airport in the best path and use backtracking to the reconstruct the route Each heap entry is wrapped in Reverse((OrderedFloat<f64>, String)) to allow the min-heap behavior

3. main.rs-
    a. The main hold is the reading and printing of the CSV, and my main function is to print out all the information.

Key Functions & Types (Structs, Enums, Traits, etc.)
Purpose:
Enum- columnval to represent the value of the dataframe cell that was either a string or a f64.
Inputs/Outputs

- The CSV values
- Place in a dataframe.columns vector

Core logic and key components

- To make one represent the origin and destination names
- And to print out the time it took to get there.

Purpose:
Struct- Dataframe to customize the data structure to represent the CSV
Inputs/Outputs

- The CSV file and the column type
- Created a memory table of CSV values with the types

Core logic and key components

- Holds the label, which is the header
- Columns, which are the row values
- And the type of either one or two

Purpose:
Two functions are in the data frame struct: the read-csv, which reads the file and fills the dataframe with type values. And print the dataframe to have it in a table format. This helps me understand how the dataframe looks when dealing with the data.
Inputs/Outputs

- To read CSV, the input is the file path and the type of vector, while the print_dataframe input was self-only
- The result is either that the file is read or an error message is displayed.

Core logic and key components

- Uses the read csv::Readerbuilder
- Create the rows according to the enum rule.
- Working to iterate through the column label and the rows, match the enum values, and print based on that information.

Key Functions
Main function :
Purpose: it leads the CSV, builds the graph based on the CSV, and then takes the user input to find the shortest route.

Inputs/Outputs
The input in the terminal is the airport codes, and the output is the pathways that work the best.

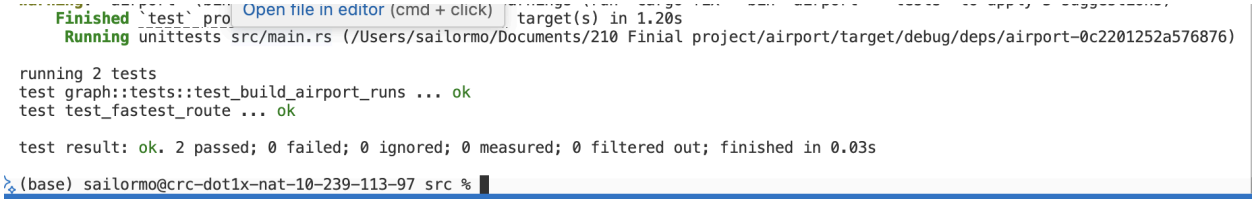
Core logic and key components

- The use of read_csv was made to create the CSV file and call the build_airport function, which converts the data frame into a graph structure. It is building the airport connection with weights.
- It will take the loop to take the start and the destination airport code
- Call the fastest route from the path mode to find the best path to take.

Main Workflow

The modules work together by first reading the CSV file in main.rs, converting it into a structured format, and then building the graph from graph.rs to process the data frame into a weighted graph with the average time and standard deviation. The user will then input what airport they want to go to, and based on their destination, the path.RSs will use the fastest_route function to find the shortest path, displaying the length of the flight and the airport path.

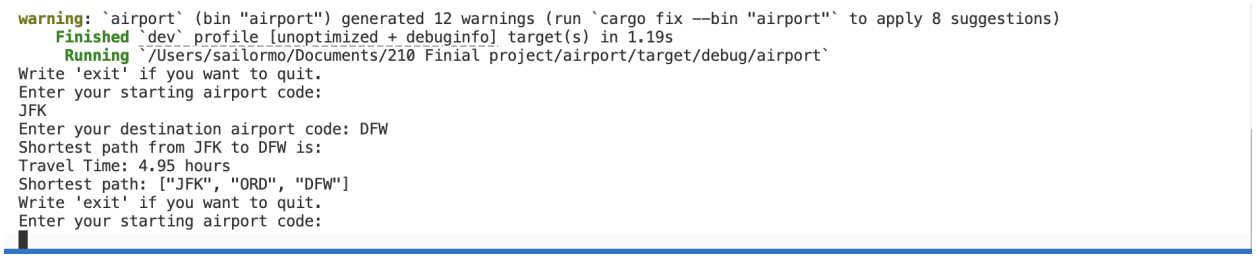
D. Tests
Cargo test output (paste logs or provide screenshots).



For each test, what it checks and why it matters.

- The first test checks if my graph is being implemented correctly by using a fake dataframe and creating my hashmap of hashmap type.
- The second test aims to identify the fastest algorithm using the CSV file information for flights from JFK to LAX.

E. Results
All program outputs (screenshots or pasted).



F. Usage Instructions
How to build and run your code.
Description of any command-line arguments or user interaction in the terminal.

- Cargo run will prompt the user to write an airport code name, eg, JFK, LAX, DFW, to find the shortest pathway and how long it will take to get there. It should only take a couple of seconds for the algorithm to produce an answer.

G. AI Usage
https://chatgpt.com/share/6813e00b-8fd4-8012-9f55-ef64db275206

Learning how to implement Dijkstra's algorithm in the assignment for the shortest path and how to fix my code, which is based on the class lecture notes. I understood the missing elements of my code when dealing with the function's structure. The conversation revealed that I need to save the path that was taken previously. Additionally, I needed to use nested HashMap logic with .get() to access the graph. I was trying to copy the syntax given in the class lecture and was not sure how to restructure that to work for a nested hashmap.

Rundown of how the code works.
The fastest_route function references the graph, start, and goal airport. It will then return an option of the average travel time and the airport code for the path taken. Then, in the while loop, go through all the paths in the queue and pop the airports with the shortest known time, and see if the goal airport was reached. The function then makes the route backward using the state goal and follows the previous map to the origin airport. Every airport found in that direction will then be pushed onto the path. Use reverse on the path to it in the correct order. If the airport has neighbors, it will hold a HashMap of destination airports and the flight information. Then, I used a for loop to iterate through the direct flights from the current airport. Using the distance and the stats information, calculate the new flight time. Then compare the new path to the neighbor, which is shorter than any other path found previously. Using the map_or method on the distance. Once the shortest path is found, update the distance in the distance map and record the path. Finally, push the neighbor and the new distance into the priority queue.

H. Additional comments:
The CSV does not track all flights and will produce a different answer for JFK to DFW and DFW to JFK because the flight information may not include a time for JFK to DFW, but may have a row with information about DFW to JFK. This dataset may also deviate from the actual flight times between airports.