Travel Planner

CS 225 Final Project FA22

Our Team

Kathryn Thompson

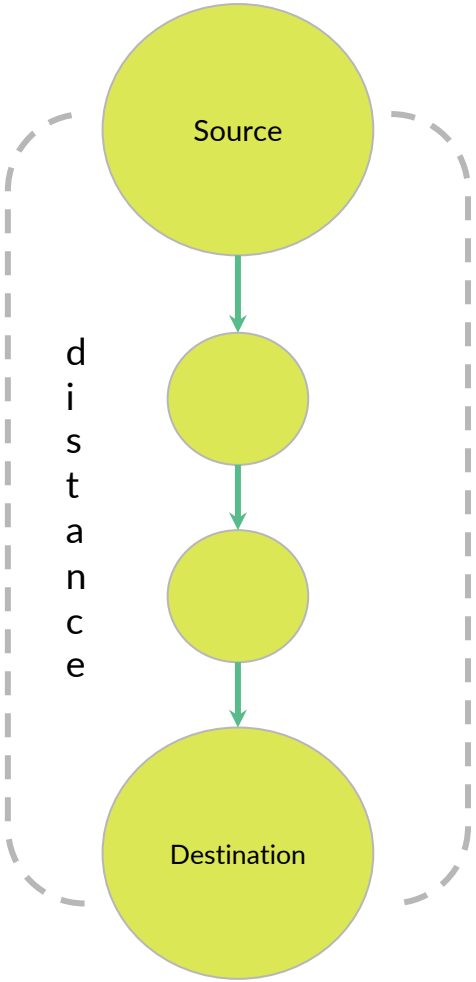
Frank Lu

Shriya Surti

Neha Valavala

1. Goals

Through this project, we wanted to create a Travel Planner that helps travelers decide where they want to go based on their current location and how far they wish to travel.



Overview

Objectives:

1. Graph Implementation
2. BFS
3. Dijkstra’s Shortest Path
4. Betweenness Centrality

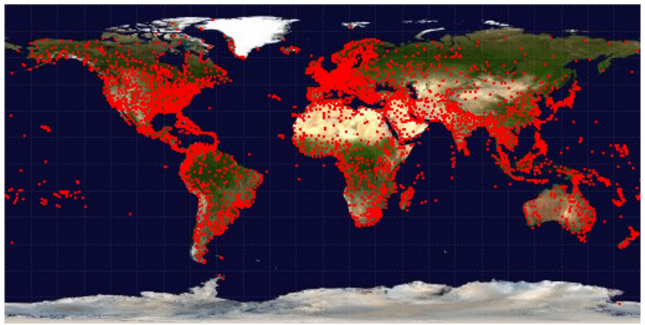
Data

From the OpenFlights Database

Airports Data

The database contains data about the airports around the world.

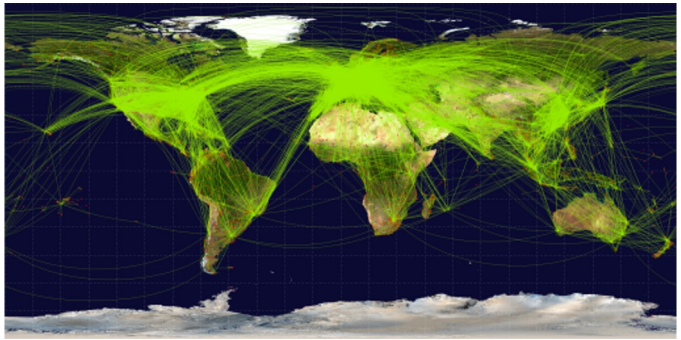
- Only used the "Airport ID", "City", "Country", "Latitude", and "Longitude" properties of each entry.



Routes Data

The database contains data about flight routes that connect airports

- Only used the "Source airport ID", "Destination airport ID", and "stops".

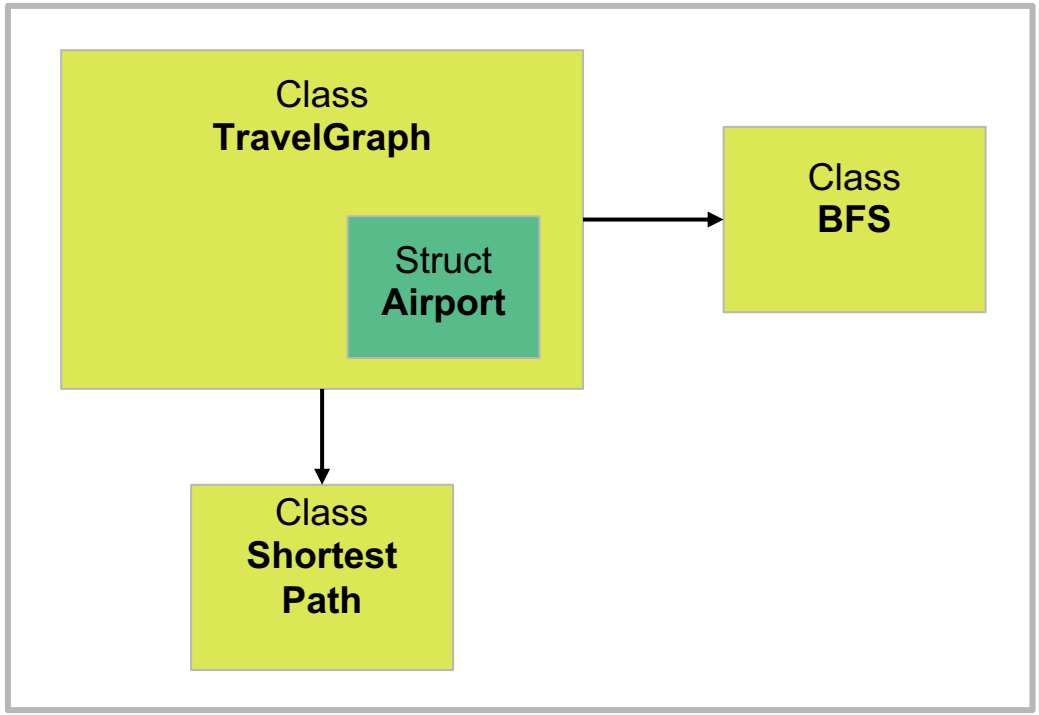


2. Development

Data Structure

Classes:

1. Travel Graph
    - Airport
1. BFS
2. Shortest



Challenges

1. Creating the Makefile
2. Planning how to implement the adjacency list
3. Test Cases and Linker Errors
4. Collaborating on GitHub
5. Planning the Travel Planner's functionality

```
g++ -Wall -g -w -c CS_Project/tests/tests.cpp
CS_Project/tests/tests.cpp:40:10: error: expected constructor, destructor, or type conversion before ‘(’ token
   40 | TEST_CASE("read data medium") {
      |          ^
CS_Project/tests/tests.cpp:83:10: error: expected constructor, destructor, or type conversion before ‘(’ token
   83 | TEST_CASE("distanceBetween() test1") {
      |          ^
```

Resolutions

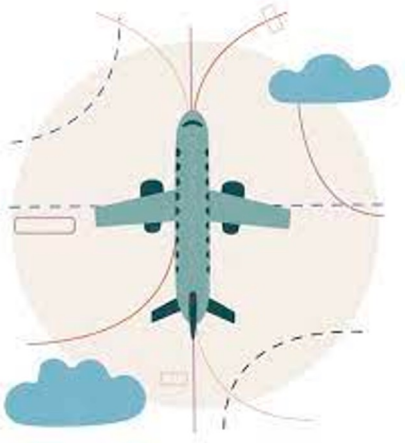
We resolved most of our issues by asking the CAs and TAs at office hours, especially errors with our test cases and Makefile.

After we were able to test our code, we had to debug, and change some of our graph implementation in order to get it to work correctly. Especially with the storing of the adjacency lists. We used a vector, but initially wanted to use a map. The problem was overloading the operator<() in our struct.

Further, collaborating over GitHub was a major challenge for us when everyone was working on code at the same time. However, with communication, we were able to resolve this issue.

Limitations

- Shortest Path
- Optimization
- Testing edge cases



3. Conclusions

Travel Graph

Description: Most of our implementation is dependent on storing our data in the right way. So, the TravelGraph constructor is one of the most important parts of out Travel Planner.

Input: Two Files- one for the airports data, one for the routes data

Output: The constructor stores our data in a graph implemented as an adjacency list

Time Complexity: O(n^2)

Application: Stores the data in a graph using adjacency lists. This is the base of the map between airports that the user uses indirectly to plan their travel

Shortest Graph (Dijkstra's)

Description: Our shortest path algorithm finds the travel path with the shortest distance between the source and destination.

Input: Travel Graph, Source Airport, Destination Airport

Output: Vector of airports and their distance from the previous airport

Time Complexity: O(V^2)

Application: In our trip planner, the user can choose to travel the shortest distance to their destination using this function

Betweenness Centrality

Description: Uses the Betweenness Centrality algorithm to find the level of connectedness of an airport

Input: Travel Graph, Airport

Output: How connected the airport is (scale of 0 to 1)

Time Complexity: O(VE + (V^2)log(V))

Application: The user can choose to go to a more ‘central’ or connected airport if they want more travel options

BFS

Description: A breadth first traversal of the Travel Graph

Input: Source Airport

Output: Vector of Airport IDs

Time Complexity: O(V + E)

Application: The BFS helps the users know which airports they can travel to from the airport that they are currently at.

Reflections

- **Learning:**
  - Apply theory we learnt in class
  - Work with and clean real life data
  - Working with a Makefile
- **Thoughts:**
  - We underestimated the amount of work required to complete the project
  - When we expected to accomplish a lot more, and only recognized the limitations later
- **Extension:**

(If we had more time...)

  - We would implement a shortest path algorithm that looks at stopovers instead of distance
  - We would the Betweenness Centrality function to allow users the choice to travel to a more connected airport
  - We would try to improve runtime and memory requirements