Team 13
Chase Wrenn, Rishi Sajay, Max LoGalbo
COP 3530
Dr. Amanpreet Kapoor
2 August 2021

Project 3 Documentation

Administrative:

Github URL: https://github.com/chasewrennufl/Project3bFinal
Video Link: https://www.youtube.com/watch?v=xrzY91VpRv8

Extended and Refined Proposal:

Problem: What problem are we trying to solve?

Mainly, we are trying to solve the problem of knowing the cheapest route between multiple places. Travel agencies can do something similar, but to find the lowest possible price within a given time of the year is a different problem entirely. Our program will find the cheapest possible route to a set of destinations given a designated airline and quarter of the year. This might not be available at the individual's time they hope, but they will know what the best deal possible could be for their travel plans.

Motivation: Why is this a problem?

Some individuals may work very strictly on a budget, and as a result, cannot afford many flights. There is not a readily accessible site to find trip data based on certain parameters, such as cost, distance, and availability. Most individuals can only find the current prices for a trip. This program will find the lowest possible price to travel to a set of destinations and back, checking to see if the trip is too expensive for the user. Our solution will optimize a trip by finding the cheapest route available to reach a list of destinations.

Features implemented

Users can choose the starting location, the quarter, the airline, and multiple destinations. All choices are provided with clickable buttons, so the user doesn't have to type anything. Once the user presses "Compute," the program computes the shortest paths with Djikstra's and Bellman Ford's traversal algorithms to provide the user with the cheapest paths from the start point to the destinations.

Description of data

The data is a compilation of various flight statistics from 2018 in the US. The original data has a variety of statistics, but for the purpose of our project, the data has been condensed from 27 million rows to 9 million rows. The data consists of flights regarding 12 airlines, 4 quarters, and 53 locations. Out of the 14 variables provided, we use miles, airline company, price per ticket, quarter, origin, and destination.

Tools/Languages/APIs/Libraries used

This project utilizes C++, as all members in this group are most familiar with the language. We used numerous C++ libraries, including input/output file stream, vectors, sets and maps. The project is compiled with Visual Studios, and XCode.

This project utilizes SFML and TGUI. TGUI is a library based on SFML. All graphical functions are implemented directly through TGUI, not SFML. SFML is just used as the base TGUI is built upon. TGUI provided a variety of graphical functions used in the project, including buttons, scrollbars, labels, and textboxes. TGUI has built in clickable buttons which was integral to the GUI portion of the program, as button pressed functions direct the algorithms to compute the shortest paths.

Data Structures/Algorithms implemented

We implemented the Dijkstra Algorithm and the Bellman Ford Algorithm to find the shortest path between multiple locations.

Data Structures/Algorithms used

We utilized sets, maps and vectors. In the main function, maps are used to store the location data and the airline data. This is used mainly to help cross reference between nodes or flights (edges) and their respective weights or adjacent edges. Sets are used to store all destinations the user wants to travel to. Within our graph of the different flights, we implemented an adjacency list to store our map. Within the Dijkstra and Bellman Ford applications, we used vectors to store our flight paths when returned. To keep track of the matrices storing our temporary previous node and distance from source node data, we used maps with the keys representing nodes to keep track of these values. In addition, to help the GUI function, we used sets to help store airlines and destinations.

Distribution of Responsibility and Roles: Who did what?
Chase Wrenn implemented the Bellman Ford search algorithm and created the flight manager and flight graph files. His algorithm was one of the two that are being compared to see the difference in computational time. Chase also helped with documentation. In addition, Chase helped lead the project and delegate tasks to members, keeping tabs with their progress.

Max LoGalbo implemented Djikstra's algorithm for the project. When the user inputs their specifications, both Djikstra' and Bellman Ford's algorithms run and are compared. Max also helped with testing and documentation as well as assisting with GitHub issues. Furthermore, he assisted with video editing and the compilation of the video.

Rishi Sajay was in charge of the graphical user interface. He utilized TGUI to create an interactive window for the user to specify their requirements for the program. The user picks a starting location, the airline, the quarter, and the destinations. The user presses the compute button and the output window shows the results of running both Djikstra's and Bellman Ford's algorithms. Rishi also helped with the documentation and testing.

Analysis:

Multiple changes were made from the proposal to the final product. The original idea was that users can input the desired price and distance and the program will output a list of feasible locations. However, this wasn't very practical as people usually know where they want to go, and what they can afford. Our new implementation lets users pick a starting location, an airline, the quarter they are travelling in, and a list of destinations. The program outputs the cheapest flights to each destination if available. Our proposal included the idea that users can input the cost of the flight based on their budget, but the way the data was structured eliminated this as a possibility. The final implementation provides the cheapest costs for the user to choose from.

Complexity Analysis:
void FileManager::buildGraph(string fileName, short q, map<string, bool> airlines, FlightGraph& fgraph): The computational complexity of the buildGraph function is simple O(Elog(V)), with E being the amount of edges in the graph, and V being the amount of unique vertices in our csv file. This stems from running through the entire csv file, with each row being a unique edge. For each edge, we need to run the map's find function, which has computational

complexity O(log(n)), with n being the number of entries in the map. So, the computational complexity becomes O(Elog(v)).

vector<FlightEdge> DjikstrasModAlgo::calculateRoute(FlightGraph *g, short source, vector<short> dest): The computational complexity of this algorithm comes out to O(D(Vlog(V) + Vlog(E))). This stems from having to run the algorithm D times for each destination. In turn, for each run of Dijkstra's algorithm, the computational complexity of each run in Vlog(v) + Vlog(E), due to having to run through every vertice, checking if each vertex was checked and running through each edge.

vector<FlightEdge> BellmanFordAlgo::calculateRoute(FlightGraph *g, short source, vector<short> dest): The computational complexity of this function in total is O(DEV), with D being the number of destinations selected by the user, E being the number of edges within the graph, and V being the number of vertices in the graph. This derives from having to run the Bellman Ford algorithm D+1 times, with each run of the algorithm consuming EV complexity.

main(): The computational complexity of the main method is O(DEV), with D being the number of destinations selected by the user, E being the number of edges within the graph, and V being the number of vertices in the graph. Since, out of all the methods, the Bellman Ford calculate route is most computationally expensive, this is the overall complexity. There are a few other variables, like the number of airlines and quarters, but these values are so small compared to the graphical methods that they aren't included in the time complexity analysis.

Reflection:

The overall experience was rewarding. This project incorporated multiple topics discussed in class and was much more comparable to real world projects than previous projects completed. However, we faced numerous challenges in completing this project.
One issue was a compatible GUI library. A large amount of time was spent searching for a proper library for visual effects. Another problem was combing out work. Each of us completed our section aptly, but combining our code was a whole other issue. We had to deal with creating a smooth transition so the main function (which implements the GUI) can run both search algorithms. Both search algorithms use the same graph structure, which greatly helped with the integration of the project. Another major issue was loading the data. For some reason, using Visual Studios took an extensive amount of time loading the entirety of the data (well over 5 minutes). However, when using cLion, the data loaded and the algorithms ran in under 20

seconds. The GUI portion of the project (made with TGUI) was built with visual studios however. Because of this, we had to figure out how to establish TGUI through XCode and run the program on XCode .

If we were to start the project again, we would immediately utilize cLion from the start. The data loads tremendously faster on that IDE as opposed to Visual Studios. Another change would be to start the project earlier. The summer curriculum was heavily accelerated, and as a result the majority of the project was built a few days prior to the deadline. Starting earlier would give us more time to combine our code and create a smoothly implemented finished product.

What we learned:
Rishi - I learned how valuable external libraries are. TGUI made the graphical user interface much more feasible to accomplish. With SFML, registering buttons and mouse clicks would have taken a tremendous amount of time, but the library allowed me to create a polished input output window for the user to clearly see what they choose and the results of running the program.

Max - I learned the importance that IDEs can have on the development process. It surprised me when two different IDEs would run the same set of code in significantly different time (by a factor of ~15). As a result, I might look into faster IDEs for my personal use. Furthermore, I learned about the art of debugging a project. Mainly, I learned the skill of testing my code as I wrote along with setting breakpoints. In turn, I was able to write the code relatively quickly and with less bugs than usual.

Chase - I learned how versatile an algorithm such as Djikstra's or Bellman Ford can be. These algorithms can be mutated and used in a variety of ways to solve a number of problems. This project showed a real world application of this. I also learned the importance of pseudocode. For the algorithms, I wrote out the code in pseudocode first, which made implementation far easier. When working with algorithms, planning in pseudocode is vital.

References:

The source of our data set: https://www.kaggle.com/zernach/2018-airplane-flights