Administrative

- **Team Name**
  - Route Optimizers
- **Team Members**
  - Reggie Segovia
    - GitHub username: rs-dkd
  - Jonas Kazimli
    - GitHub username: jkazimli
- **Link to GitHub repo**
  - https://github.com/rs-dkd/P3-RouteOptimization/tree/main
- **Link to Video Demo**
  - https://www.youtube.com/watch?v=lsK-I8VYx6U

Extended and Refined Proposal

- **Problem:** What problem are we trying to solve?
  - We are trying to solve the problem of traffic congestion by developing a system that can predict and adapt to traffic conditions by finding the most efficient path to get from one coordinate location to another based on distance.
- **Motivation:** Why is this a problem?
  - Traffic congestion is a critical issue in urban areas as it leads to increased travel times, higher fuel consumption, and air pollution.
- **Features**: When do we know that we have solved the problem?
  - The system accurately analyzes traffic data provided.
  - It offers routing suggestions based on two different pathfinding algorithms (A-star and Dijkstra's), which can be compared for efficiency and accuracy.
  - Turns graph data into a visual result that the user can interact with and utilize to get the most optimal path and determine how it was calculated by the algorithm.
- **Description of Data**: (Public data set used and the link to the public data set) **or** (Schema of randomly generated data - i.e. what are the different columns in our dataset and the respective datatypes)
  - The primary data set we will use is OpenStreetMap data accessed through the BRIDGES API.
    - The BRIDGES API allows for direct access to OpenStreetMap data, which we are able to obtain and implement directly into our program.

        - https://bridgesuncc.github.io/datasets.html
- **Tools:** Programming languages or any tools/frameworks used
    - C++ for backend and algorithm implementation.
    - BRIDGES-C++ for implementing the OpenStreetMap dataset and creating visualizations.
- **Algorithms implemented**
    - Data Structures: Graph structures representing the city's road network, using adjacency lists for efficient pathfinding.
    - Algorithms:
        - Dijkstra's Algorithm: Finds the shortest path from a single source to all other nodes in a graph.
        - A* Algorithm: Finds the shortest path from a single source to a single target node, using heuristics to speed up the process.
- **Additional Data Structures/Algorithms used:**
    - Algorithms:
        - BFS Algorithm: Graph traversal algorithm that was used to determine if a path can be created from the starting point to the goal point that was selected by the user. This algorithm uses an unordered set to keep track of the already visited vertices/nodes and a queue to keep track of the ones yet to be explored.
- **Distribution of Responsibility and Roles:** Who is responsible for what?
    - Reggie
        - Incorporated the Bridges API to use OpenStreets map data to visualize and compare the pathfinding algorithms.
    - Jonas
        - Worked on the implementation of two algorithms (base code).

Analysis

Any changes the group made after the proposal? The rationale behind the changes.

- We decided to use Bridges API to integrate the data set and compare two algorithms. In our initial proposal, we had two options for the data set integration- SNAP from Stanford and OpenStreetMap. The main motive behind this selection was the documentation OpenStreetMap offered.
    - SNAP dataset was only for three cities. However, there were countless cities available with the OpenStreetMap (over 100 cities).
- We also decided to not use real-time traffic data, as this would have required large volumes of complex data that we would not have been able to gather and format in such a short time. To give accurate traffic representation we would have needed to utilize a data set that included and updated that data, which was not provided in the Bridges OpenStreetMap data which we decided to use.

Big O worst-case time complexity analysis of the major functions/features you implemented

'dijkstra' method in Algorithms.cpp:

- All vertices are initially set to infinity except the start node. This step is O(V). V is the number of vertices.
- Each vertex can enter the priority queue multiple times as distances are updated. The push and pop operations of the queue are O(logV) each.
- As for edges (E), every adjacent vertex to a current vertex is checked. If every vertex is connected to every other vertex this step can occur O(E) times. And each update will take O(logV) because of the priority queue.
- In conclusion, this method will have a time complexity of O(V+ElogV).

'aStar' method in Algorithms.cpp:

- Both gScore (cost from start to node) and fScore (estimated total cost from start through the node to goal) are initialized for all nodes except the start node. Therefore, the complexity of this step will be O(V).
- fScore: Nodes are processed based on their fScore, which combines actual travel costs. Each queue operation (push/pop) is O(logV).
- After the above step, each adjacent node's tentative score is calculated, and if it is lower than the existing score, the priority queue is updated. This takes O(ElogV) complexity.
- In conclusion, this method will have a time complexity of O(V+ElogV).

Other methods in RouteOptimization.cpp

- mapCenter: This method calculates the center of the map using the average of coordinates. It iterates over all vertices once. Therefore, it will have O(V) time complexity.
- findClosestVertex: This method finds the closest vertex to a given coordinates by visiting each vertex. Therefore, this method will have a time complexity of O(V).
- canReach: This method uses a simple BFS to check reachability from the start node to the goal node. Since all vertices and edges are traversed, the time complexity will be O(V+E).

Reflection

As a group, how was the overall experience for the project?
Overall it went well along with some challenges. We lost a team member in the middle of the project duration. We had to do more individual work to cover for the loss. As a team of two, we explored tools and concepts that we weren't very familiar with before. The project helped us gain more knowledge on developing almost an end-to-end application.

Did you have any challenges? If so, describe.
The process of setting up the API caused us to have to go over most of the documentation for the different Bridges files. When first attempting to install the Bridges folder to work with the project, a lot of the header files in the newest version had errors that prevented the code from functioning. This led to us having to backtrack to a previous version of Bridges API to make sure that it could work without issues for our project. Version conflict between documentation and actual library for header files. Due to a difference in operating systems between the two project members, this led to a conflict in the initial set-up as the steps taken to set up the environment properly were different for both cases.

If you were to start once again as a group, any changes you would make to the project and/or workflow?

Initially, our scope for the project was large in terms of implementation, as we expected to use different features to incorporate the visualization and the interaction with the data. It would have been helpful to have a further understanding of the way that the API that we chose to import the data from works and learn that we could have implemented the visualization directly with what we used to implement the OpenStreetMap data.

Communication – sync meetings, and push our progress to GitHub more often instead of pushing it as a final product. Although it was sufficient for this project to communicate things through text, there can be issues that are created by working on things all the way through, rather than making small commits so that everyone can be up to date with changes and can work together to overcome challenges. This is particularly important when working in bigger groups, as when multiple people are working on the same project oftentimes conflicts can arise if some changes are not properly mergeable.

Comment on what each of the members learned through this process.

Jonas – real-life implementation of the two algorithms. Bridges API and how to integrate APIs into your application. Gained a further understanding of the way that A-star and Dijkstra's algorithms compare and their efficiency in pathfinding.

Reggie – I learned about how these two algorithms are incorporated into optimizing pathfinding, the differences between the two, and how visualizations can help create a clearer image of what they do. I also learned how to use libcurl and the Bridges API to create a seamless experience between visualization and different data structures.