Project 2a Proposal

Team Name
PathFinders

Team Members
- Aditya Chhabria
- Aayudesh Kaparthi
- Ahmad Hasan

Project Title
GatorPath

Problem
What's the best way to find the shortest walking path between two spots on the UF campus? Our project will figure this out by building and comparing two famous pathfinding algorithms: **Dijkstra's algorithm** and **A* Search.**

Motivation
Getting around a big campus can be tough. We all use apps like Google Maps, but how they actually work is a mystery to most people. This project will build a tool that not only finds the best path but also shows *how* it's found. We want to compare a basic algorithm (Dijkstra's) that checks every possible sub-location with a "smarter" one (A*) that determines its sub-locations through educational guesses to find the path faster. This will let us see how these computer science ideas work in real life.

Features
We'll know our project is a success when our website can:
1. Show a map of the UF campus.
2. Let a user type in a **start and end location.**
3. Run our two pathfinding algorithms (Dijkstra's and A*) to find the best route.
4. Draw both paths on the map using different colors so we can compare them.
5. Show a simple report card that compares the two algorithms on **path length, time taken,** and **how many spots they checked.**

Data
We will use real-world map data from **OpenStreetMap(OSM),** which is like a free, public version of Google Maps.
- **Public Data Set Link:** https://www.openstreetmap.org (We'll grab the data for Gainesville, FL).
- **Data Plan:** We'll write a simple script to pull out the information we need and put it into two .csv files.
    - **nodes.csv:** A list of all the intersections and important spots on campus. We will have over 100,00 of these.
        - node_ID: A unique number for each spot.
        - latitude: The GPS latitude.

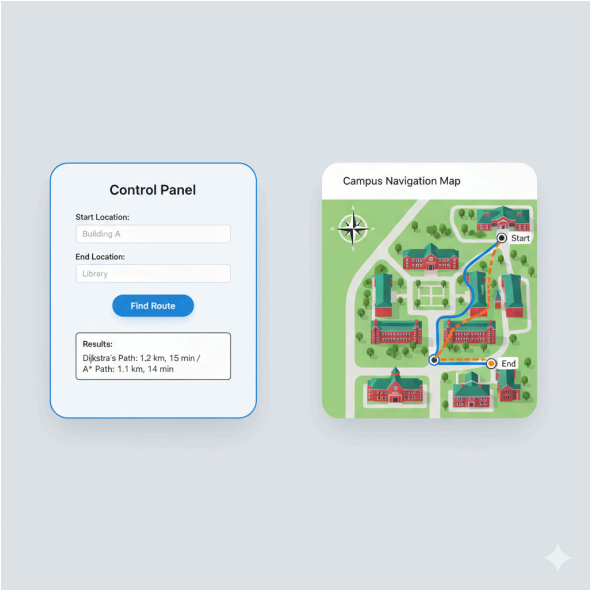
        - longitude: The GPS longitude.
    - **edges.csv:** A list of all the sidewalks and paths that connect the spots.
        - source_node_ID: The spot where a path starts.
        - destination_node_ID: The spot where a path ends.

Tools

- **Backend & Algorithms: Node.js** with **JavaScript.** The core algorithms will be written from scratch directly in JavaScript.
- **Web Server: Express.js** (A simple framework for our Node.js server).
- **Website Frontend: React** (A modern library for building user interfaces).
- **Map: Leaflet.js** (A simple and popular library for showing maps on websites).

Visuals

Our project will be a single-page website that is easy to use.



The picture above shows our plan:

1. **Control Panel(Left):** This area will have boxes to type in the start and end locations, a “Find Route” button, and a simple results box that pops up with the final comparison.
2. **Map(Right):** A map of campus. After you search, we'll draw two lines on it: a blue one for Dijkstra’s path and an orange one for A*’s path.

Strategy

Our plan is to build the entire application using JavaScript, making it easier to manage.

- **Data Structure:** We will build a graph in JavaScript to represent the campus map. We’ll use an **adjacency list** (using JavaScript Objects or Maps) to do this from scratch. This is a very efficient way to store map data.
- **Algorithms:** We will code these two algorithms from scratch in **JavaScript** on our Node.js server.

    - **Dijkstra's Algorithm:** This is the classic, reliable method. It works by always checking the closest intersection it hasn't visited yet. We will build our own **min-priority queue** class to help it do this efficiently.
    - **A* Search Algorithm:** This is a smarter, faster version of Dijkstra's. It uses a "best guess" to decide which intersection to check next. Our guess will be the **Haversine distance** (a fancy term for the straight-line "as the crow flies" distance) to the destination.

Distribution of Responsibility and Roles

- **Aditya Chhabria(Data & Backend Setup):** Will be in charge of getting the map data from OSM and writing the JavaScript code to load it into our graph structure. He will also set up the basic Node.js and Express.js server.
- **Aayudesh Kaparthi (Algorithms & API):** Will write the JavaScript code for the **graph, min-priority queue, Dijkstra's,** and **A* search** algorithms from scratch. He will also build the server API that the website will talk to.
- **Ahmad Hasan (Frontend & Visualization):** Will build the **React website** that users interact with. He will be responsible for setting up the **Leaflet.js map** and drawing the final paths for users to see.

References

- Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
- Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*.
- OpenStreetMap contributors. (2025). *Planet dump retrieved from https://planet.osm.org*.
- Leaflet.js Documentation. Retrieved from https://leafletjs.com/.