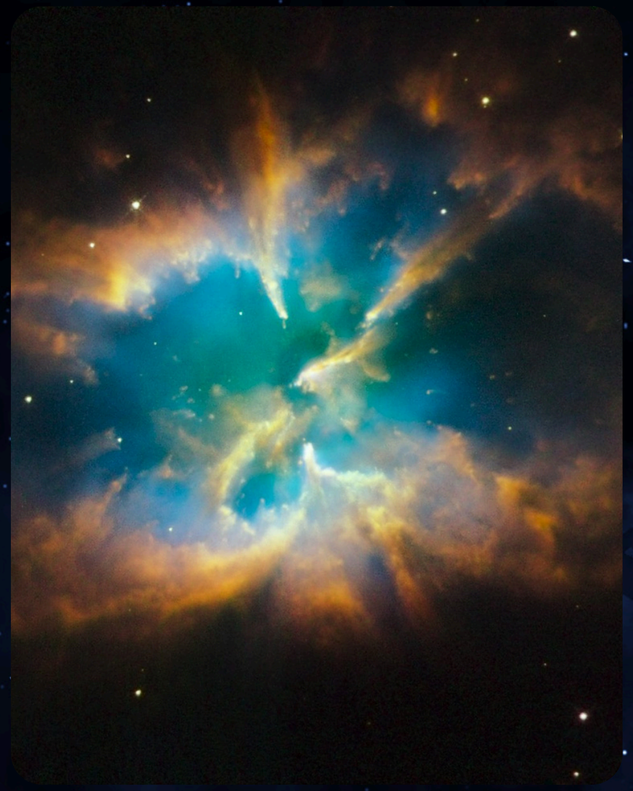
Rishabh Sankar Gaurav B Nikhil Namboodiri Nived S Mohan

PATH FINDING IN 3 DIMENSIONS

Introduction



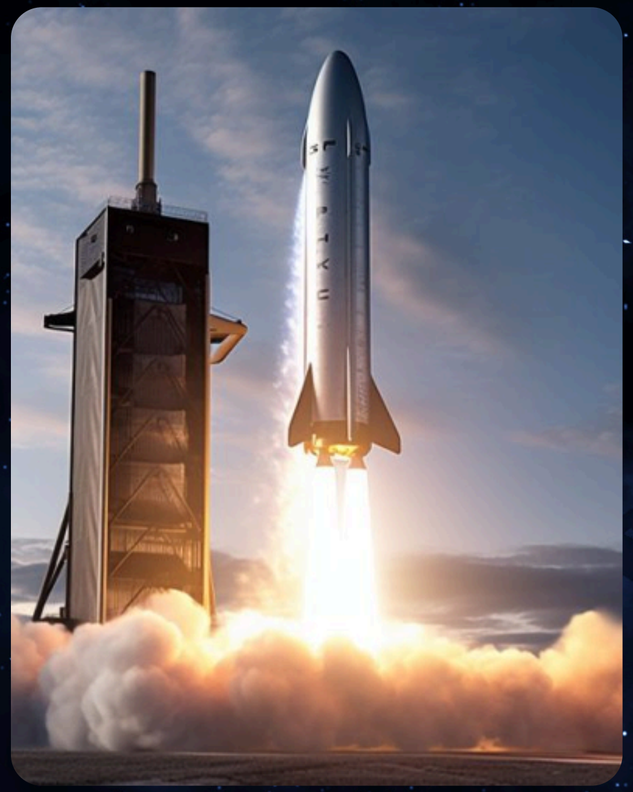
The Global Positioning System (GPS), developed by the U.S. Department of Defense, revolutionized navigation by providing civilians with real-time route guidance, eliminating the need for cumbersome paper maps and compasses. Today, GPS remains a vital tool, allowing individuals to navigate unfamiliar terrain and assisting rescue operations by offering real-time geographic updates during natural disasters.

Building on this concept, our project aims to develop a partial pathfinding system using a detailed, 1:1 map of the Milky Way. This system will dynamically update, providing real-time navigation within the galaxy.

The Environment

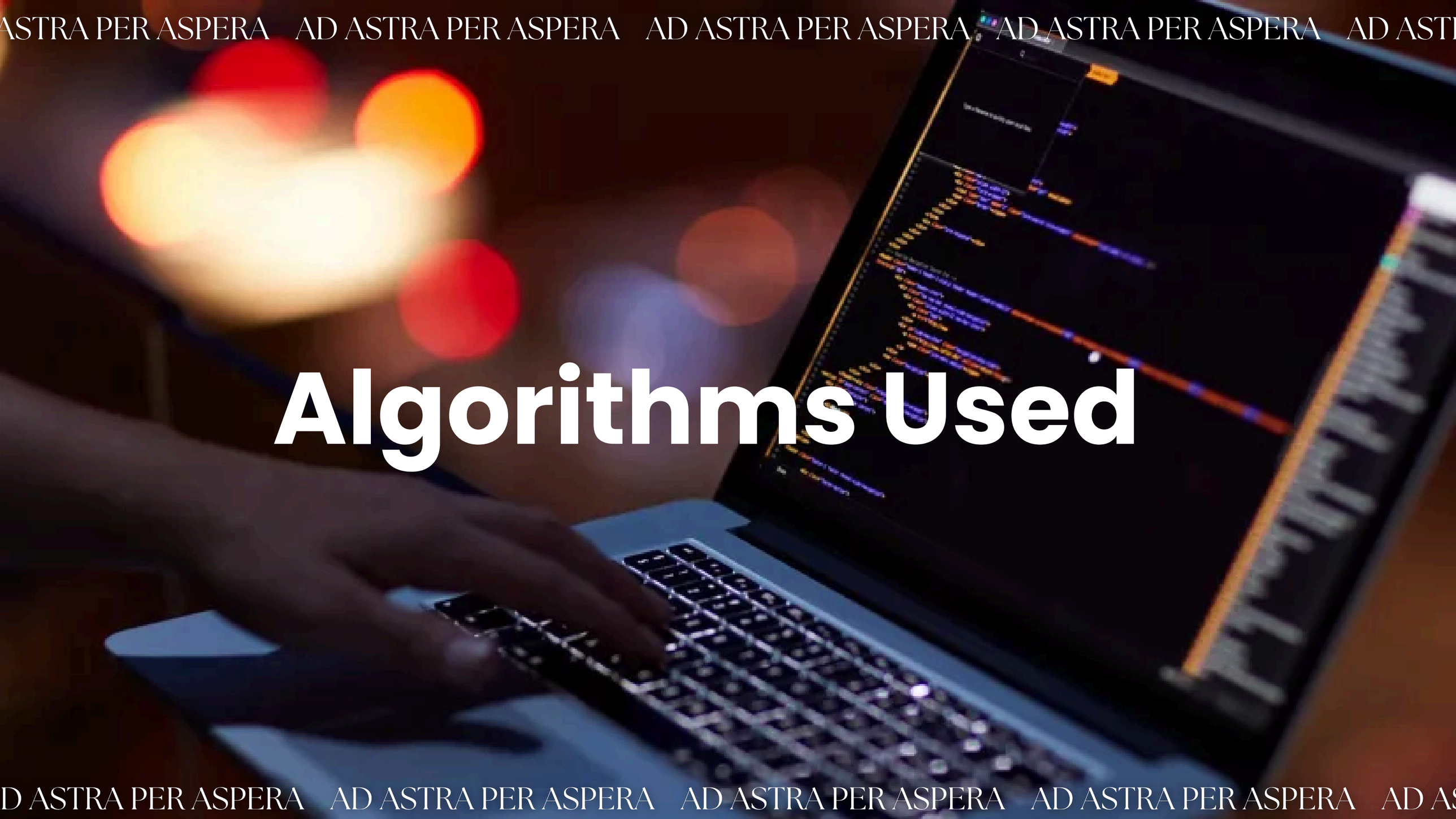
To implement a pathfinding algorithm, we needed both data and a platform. Since humanity hasn't ventured beyond the solar system, we turned to Elite Dangerous for its vast, scientifically accurate galaxy engine, Stellar Forge, which uses real data from over 160,000 star systems. Set in a future with faster-than-light (FTL) travel, Elite Dangerous offers the perfect environment to test our algorithm's effectiveness in star navigation.

Thought Process



For the pathfinding algorithm, we decided to use two approaches: Dijkstra's and A* algorithms. We chose these two algorithms to contrast uninformed and informed searching. Since A* is a heuristic algorithm, we can use a cost function that takes various factors into account to provide the algorithm with the ability to make informed decisions.

Using two algorithms also provides us with the opportunity to identify inefficiencies by providing a baseline against which we can measure the algorithm's performance, which is important for evaluating our heuristics.



Algorithms Used

Linear Search Vs Binary Search

In our map data, we only had vertex information and no edges. This led to us having to calculate edges to every neighboring star from the current star, which requires us to fetch the coordinates of every neighboring star to calculate edges.

This required over 2000 searches and more per iteration , which is a very demanding process. We first started out with linear search, expecting linear time to be fast enough for our requirements. However, it turned out to be very computationally slow.

Cont...

As we were looking for ways to decrease the time taken b the algorithm, we suddenly had an eureka moment. Since in the map data that we sourced, the id's of the star systems are sorted in ascending order, Binary Search would be the best way to retrieve the coordinates due to it halving the search space every iteration.

However this only solved one of our problems as we moved on the implementation of Dijkstra's Algorithm.

Implementation Of Dijkstra’s Algorithm

For our initial algorithm iteration, we implemented Dijkstra’s Algorithm to understand the fundamentals of pathfinding, as A* is essentially Dijkstra's with a heuristic.

We tested it by plotting a route between Sol and Deciat, which are 131.5 light years apart. To our surprise, Dijkstra’s found the optimal path in about 2 minutes, matching the in-game plotter's results. This success raised our expectations for A* to perform even better.

Working Of Dijkstra's Algorithm

In our Dijkstra's implementation, we had to modify it a bit due to the unique structure of ourdata. Since we only had vertices and no edges, we needed to find neighbours by setting a bound, which in our case was the maximum distance a ship could jump.

With the above bound, we only considered neighbours of the current nodes whose distances were less than our bound, which is why we needed to minimize calculation time as much as possible. The other difference was we only pushed a neighbour to the queue only if it wasn' t visited, but due to the denseness of our graph, we had to make an additional check if the neighbour was also not in the priority queue.

Implementation Of A* Algorithm

Now, we had to come up with a cost function so that we could convert our Dijkstra's implementation to A*. Our cost function is pretty simple. The factors used in the cost function are :

- Fuel Cost - The fuel taken to jump to a new system.
- Star Class - Penalty factor for whether refueling is possible at the next star.

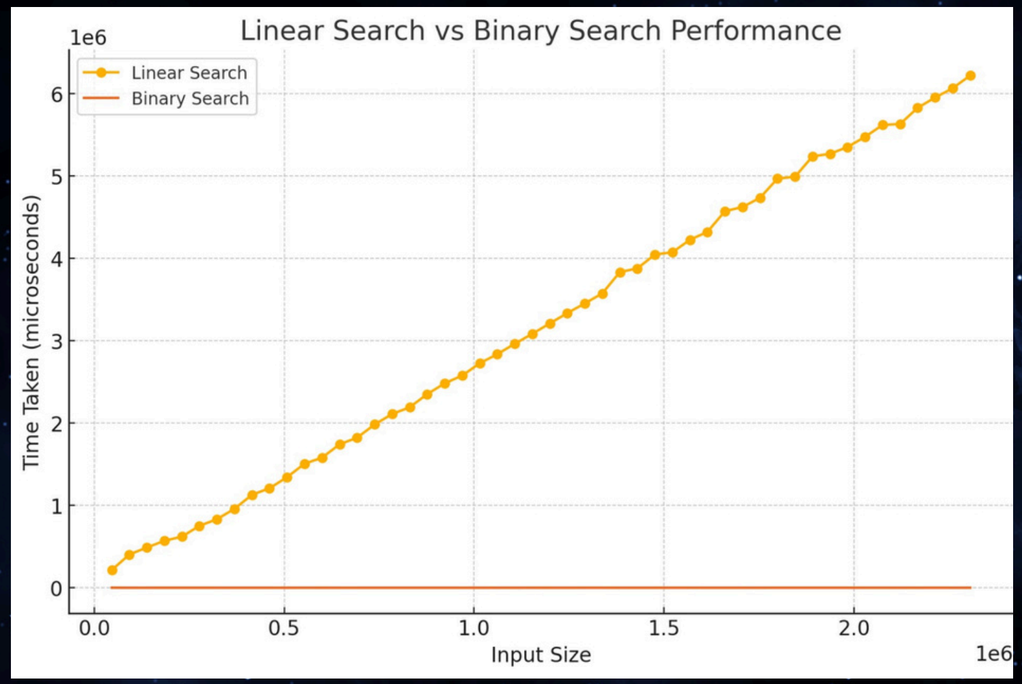
The most surprising part of this implementation was the time saved. Remember how Dijkstra's took 2 minutes to calculate a route length of 131.5 LY? For the same route, our A* implementation took roughly 8 seconds to compute the path. This was a testament to the effectiveness of the heuristic function, leading to a 94% improvement in efficiency.

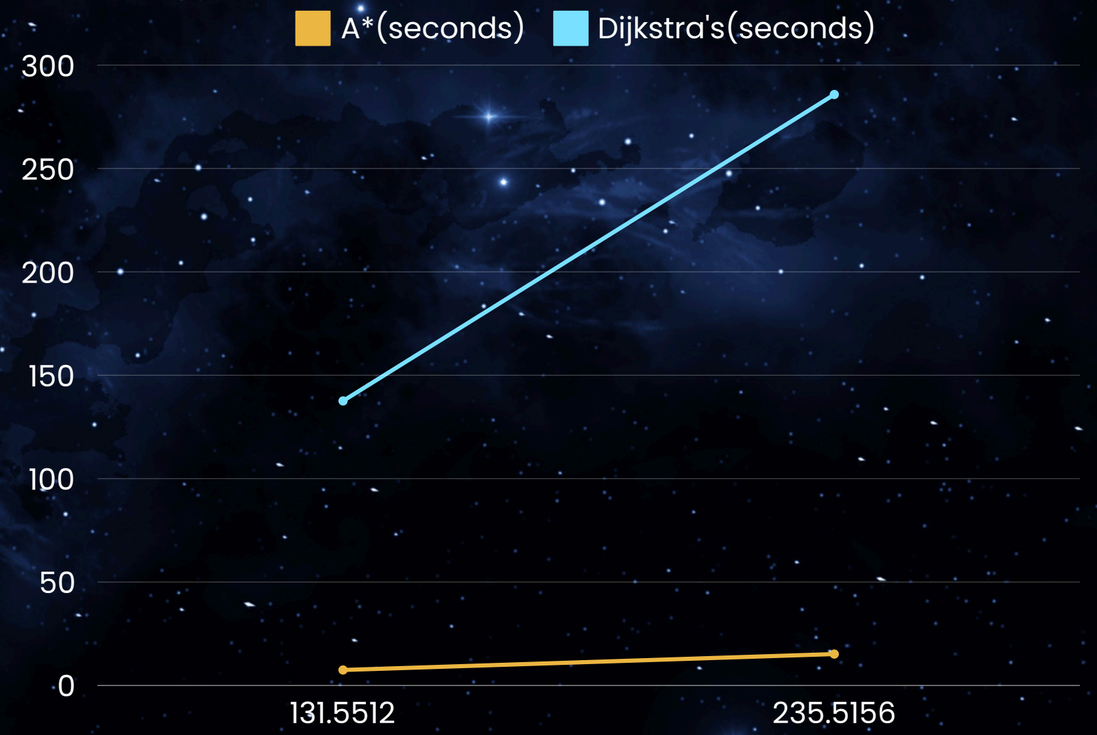
Challenges Faced

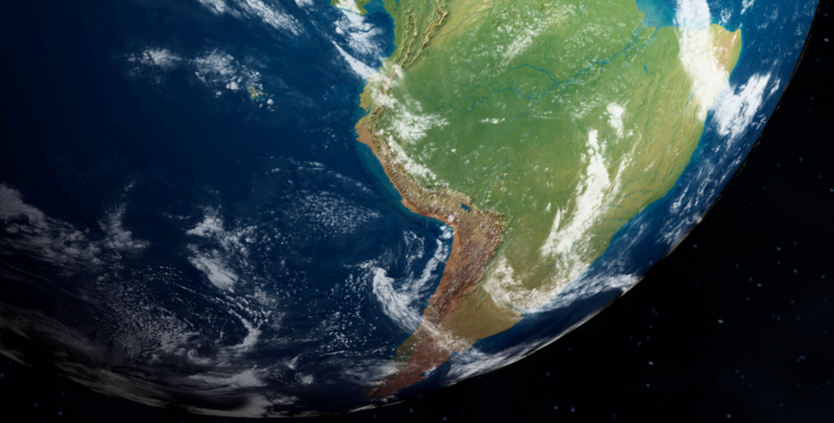
- Initially, Dijkstra's was taking a very long time and did not even compute the path. After debugging, we realised that neighbours that were already processed were being added to the queue which was extremely sub-optimal.

- When we were calculating euclidean distances, we had a very strange issue. For some reason, distances between the goal node and current node incremented every time instead of decrementing, which led to indeterminate results, as the algorithm never terminated even after 10000 iterations.

Graphs







THANK YOU

THANK YOU