

INNOVATION LAB'20

Name : Aman Gupta

Roll No. : 1801CS05

Date : 20/05/2020

Eco-Routing

Brief to what I've done so far.

- Learned Python.
- Learned OSMNX.
- Learned and applied various inbuilt functions of OSMNX on our campus and various locations of Patna city.
- Before mid-sem, I wrote various OSMNX code for the given algorithm to find a route for EV.
- After mid-sem, written algorithm in C++, some of them are :
 - K-Shortest Path for EV.
 - Shortest Route passing through particular points.
 - Shortest Path for EV when it has to charge in between source and destination considering vehicle can charge at any node.
 - Shortest Path for EV based on Distance and Time considering some Charging Station at a given location including the location where it has to charge.
 - Added an interactive way to show route and charging stations within the graph using folium.
 - Shortest Path for EV based on Distance and Time considering some Charging Station at a given location including the multiple locations where it has to charge.

Assumption for the latest algo attached with mail are :

Range of vehicle $\geq 5000\text{M}$ (15KM). (As charging stations are not optimal)

Radius of City $\leq 5000\text{M}$ (5KM). (Depends on system and internet connection)

Considered 50 charging stations at random locations within the given graph along with random waiting time.

Jupyter Notebook :

Snippet 1 :

Taking coordinates of location and radius of the location.

Snippet 2 :

Generating a graph in terms of nodes of that location which we have taken as input.

This will also generate an 0.html in which we can see all nodes, along with nodes we can also see 50 different charging stations in red colour.

It will also generate a .txt file in which we will have a count of nodes along with roads between nodes and their length.



Here we can see all nodes of the graph.



Here we can see the charging station in blue colour.

Snippet 3 :

Taking inputs such as source, destination, initial charge, unavailable nodes and range of our electrical vehicle.

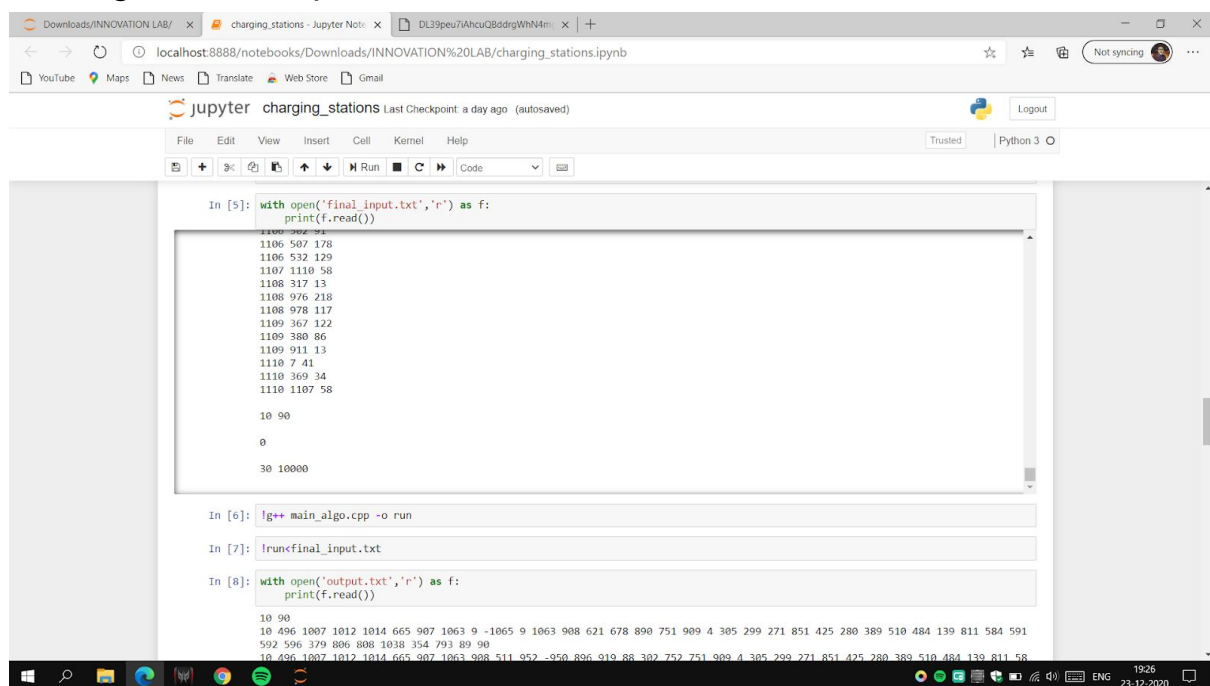
It will also generate a .txt file with all the input data.

Snippet 4 :

This code is just combining the both files, which we generated in snippet 2 and 3. It will generate and final input we will give as input to our main_algo.cpp as input.

Snippet 5 :

Printing out final input.



The screenshot shows a Jupyter Notebook window titled 'charging_stations'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Help), a toolbar with icons for file operations and execution, and a status bar at the bottom indicating 'Python 3'. The notebook contains three code cells:

```
In [5]: with open('final_input.txt','r') as f:
        print(f.read())
```

The output of cell [5] is displayed in a scrollable area:

```
1100 900 90
1106 507 129
1106 532 129
1107 1110 58
1108 317 13
1108 976 218
1108 978 117
1109 367 122
1109 380 86
1109 911 13
1110 7 41
1110 369 34
1110 1107 58

10 90

0

30 10000
```

```
In [6]: !g++ main_algo.cpp -o run

In [7]: !run final_input.txt

In [8]: with open('output.txt','r') as f:
        print(f.read())
```

The output of cell [8] is displayed below:

```
10 90
10 496 1807 1012 1014 665 907 1063 9 -1065 9 1063 908 621 678 890 751 909 4 305 299 271 851 425 280 389 510 484 139 811 584 591
592 596 379 806 808 1038 354 793 89 90
10 496 1807 1012 1014 665 907 1063 908 511 952 -950 896 919 88 302 752 751 909 4 305 299 271 851 425 280 389 510 484 139 811 58
```

The Windows taskbar at the bottom shows the system clock as 19:26 on 23-12-2020.

Snippet 6 :

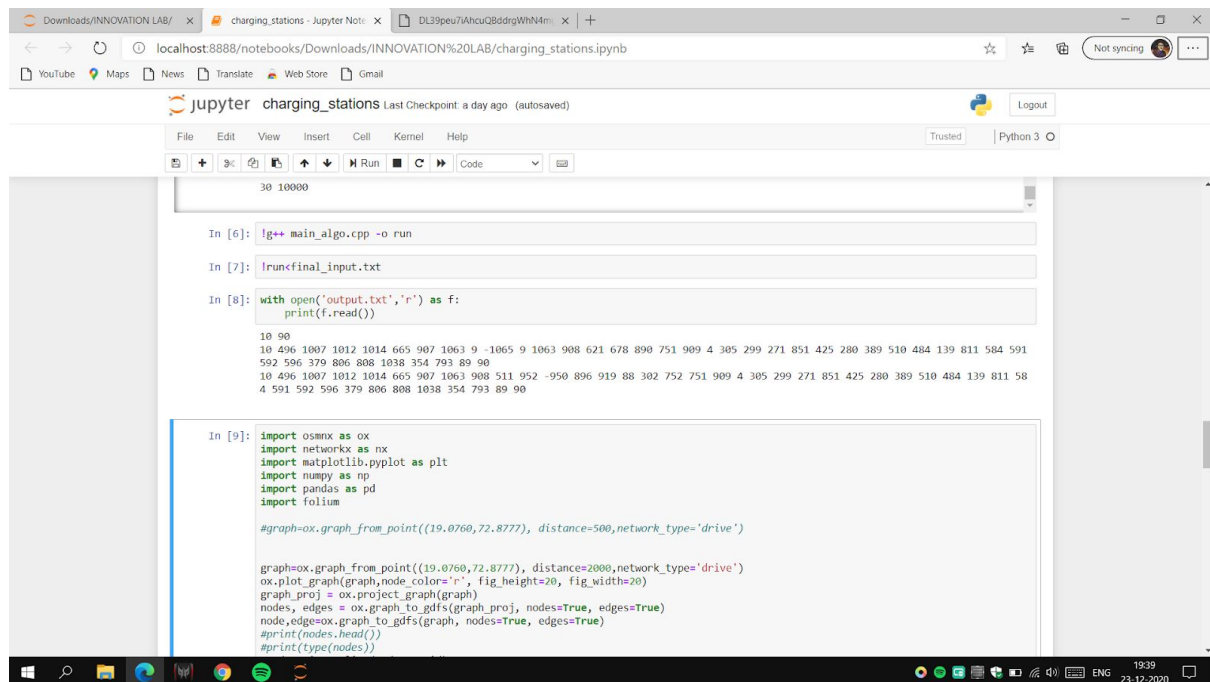
Calling our main_algo.cpp to execute and create an .exe file.

Snippet 7 :

Finally running our .exe file with final_input.txt which we have generated in snippet 4.

Snippet 8 :

Printing out optimal paths, one will have least possible time whereas other will have least distance possible.



```
In [6]: !g++ main_algo.cpp -o run

In [7]: !runfinal_input.txt

In [8]: with open('output.txt','r') as f:
        print(f.read())

10 90
10 496 1007 1012 1014 665 907 1063 9 -1065 9 1063 908 621 678 890 751 909 4 305 299 271 851 425 280 389 510 484 139 811 584 591
592 596 379 806 808 1038 354 793 89 90
10 496 1007 1012 1014 665 907 1063 908 511 952 -950 896 919 88 302 752 751 909 4 305 299 271 851 425 280 389 510 484 139 811 58
4 591 592 596 379 806 808 1038 354 793 89 90

In [9]: import osmnx as ox
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import folium

#graph=ox.graph_from_point((19.0760,72.8777), distance=500,network_type='drive')

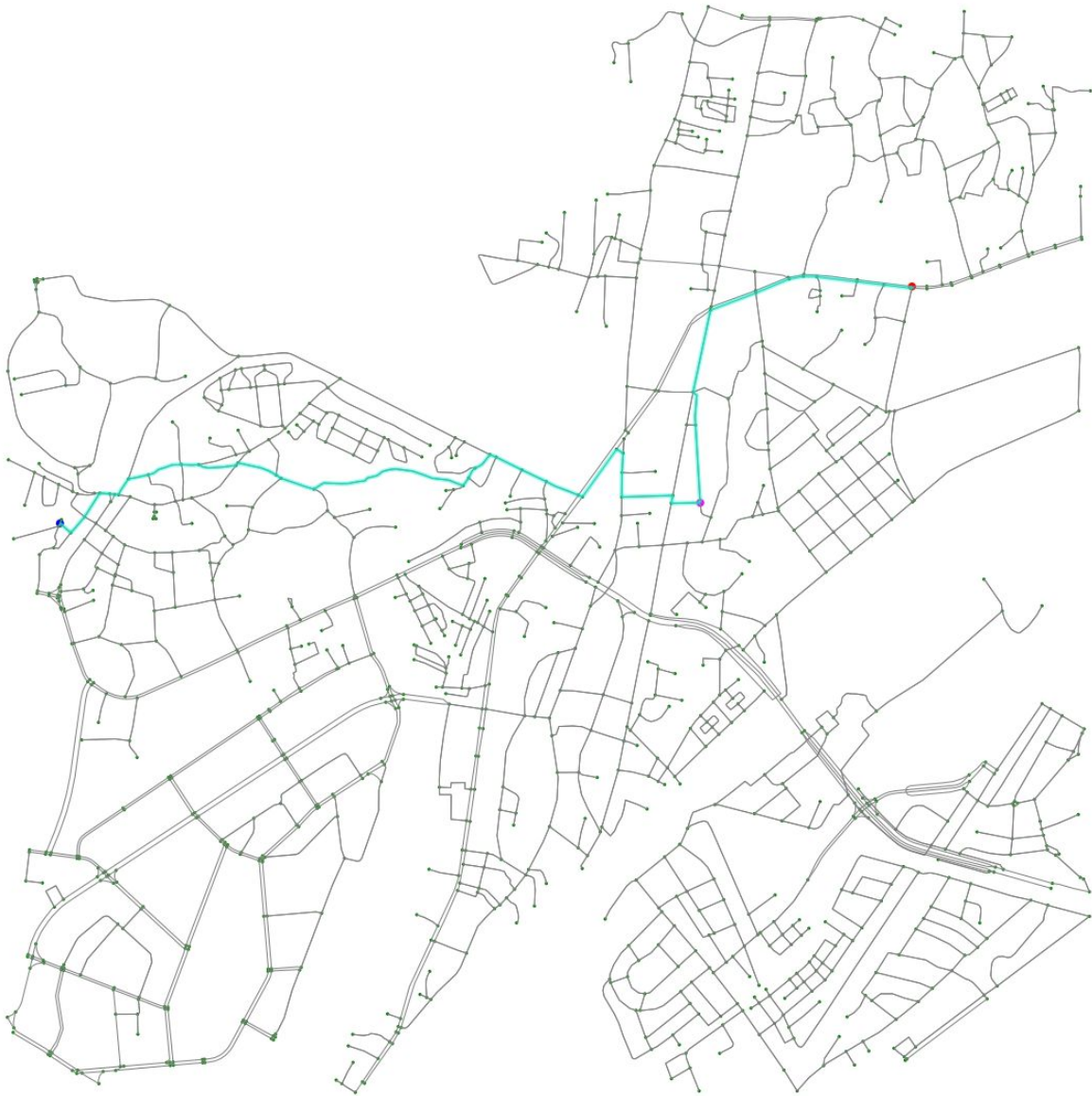
graph=ox.graph_from_point((19.0760,72.8777), distance=2000,network_type='drive')
ox.plot_graph(graph,node_color='r', fig_height=20, fig_width=20)
graph_proj = ox.project_graph(graph)
nodes, edges = ox.graph_to_gdfs(graph_proj, nodes=True, edges=True)
node,edge=ox.graph_to_gdfs(graph, nodes=True, edges=True)
#print(nodes.head())
#print(type(nodes))
```

Snippet 9 :

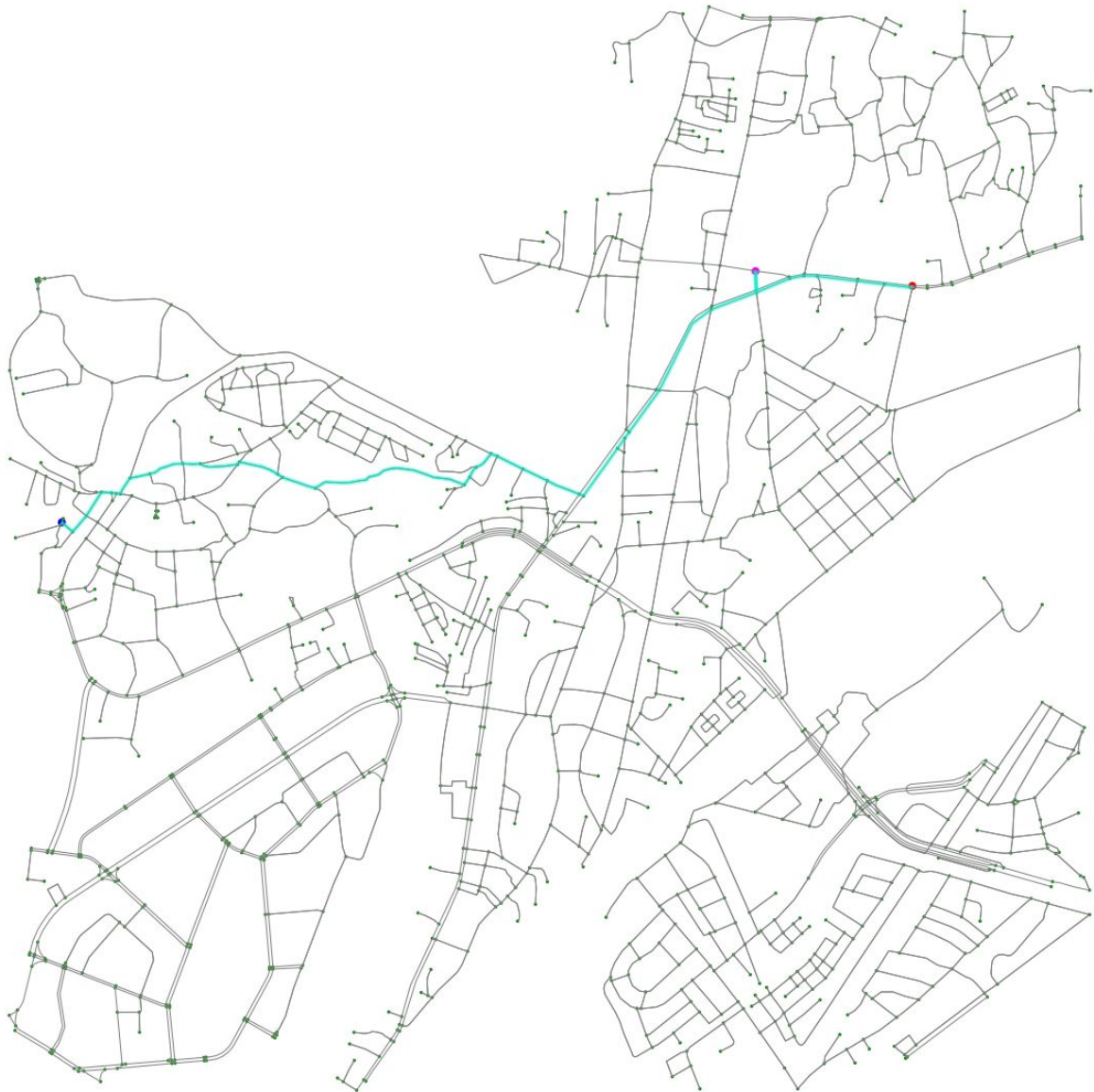
The most important snippet which will print out paths for the user in .html format using the folium library.

We are initially reading our path we have stored from snippet we also have stored our charging stations location where we will have to charge along with the least possible percentage of charge, we also have total travel time and total distance which we will print on destination node, how I generated all this will be explained in main_algo.cpp section at the last of this pdf.

Here is the sample how this all looks like :

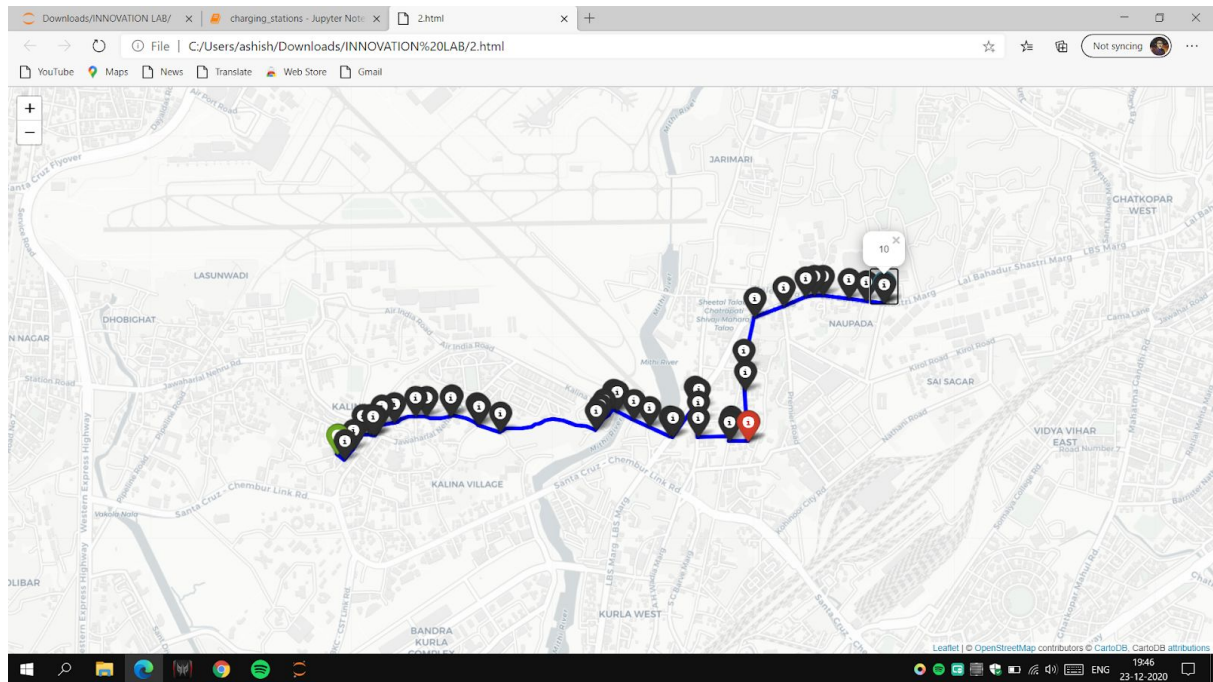


This is the path which will minimize the possible time to travel. Red node is the source and the blue one is the destination. Purple node is the charging station where vehicles should charge in order to complete their journey in minimum possible time.

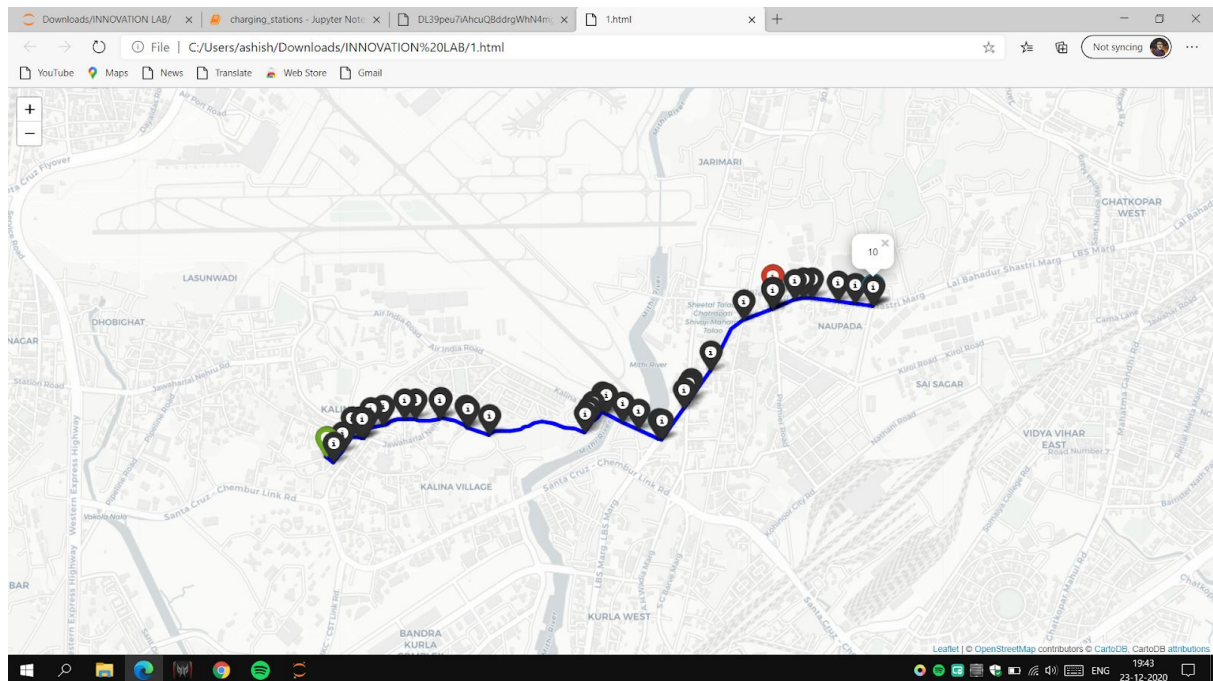


This one shows the path which will take minimum distance in order to complete the journey.

Here is how it looks on the web.



Red node is the charging station, green is the destination and blue one which shows 10 is the source of journey. This one shows the path with the minimum possible time.



This one is for minimum distance.

main_algo.cpp :

My algorithm is the modification of dijkstra in which I am running a brute approach to check on which charging station we need to charge and by how many charging stations we need to charge. This also provides the time of journey.

For a detailed description I am adding comments in my code.